Ouriginal by Turnitin

Document Information

Analyzed document	DBMS.pdf (D166432388)
Submitted	5/10/2023 10:30:00 AM
Submitted by	Mumtaz B
Submitter email	mumtaz@code.dbuniversity.ac.in
Similarity	5%
Analysis address	mumtaz.dbuni@analysis.urkund.com

Sources included in the report

SA	**248E1130_RDBMS.docx** Document 248E1130_RDBMS.docx (D165247738)	28
SA	**BOOK SIZE.docx** Document BOOK SIZE.docx (D50092775)	48
SA	**Assam Don Bosco University / Advanced DBMS.pdf** Document Advanced DBMS.pdf (D166063498) Submitted by: mumtaz@code.dbuniversity.ac.in Receiver: mumtaz.dbuni@analysis.urkund.com	42
W	URL: https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf Fetched: 1/11/2023 8:18:44 AM	30
SA	**DBMS_AIML_FINAL.pdf** Document DBMS_AIML_FINAL.pdf (D111167082)	14
SA	**MCA-Sem-II-Database Management System.pdf** Document MCA-Sem-II-Database Management System.pdf (D143652738)	29
SA	**Database System.pdf** Document Database System.pdf (D165747767)	7
SA	**47F417BA15858476660.pdf** Document 47F417BA15858476660.pdf (D123781188)	23
SA	**DBMS.docx** Document DBMS.docx (D68067493)	5
SA	**all in one.pdf** Document all in one.pdf (D109420358)	1
W	URL: http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf Fetched: 11/12/2021 9:06:00 AM	16

W	URL: https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288 Fetched: 11/21/2022 1:43:33 PM	15
W	URL: https://present5.com/database-theory-jason-fan-outline-basic/ Fetched: 9/21/2021 3:53:49 PM	20
SA	**DCAP 011.docx** Document DCAP 011.docx (D142453284)	6
SA	**Unit-4 (Part-II).docx** Document Unit-4 (Part-II).docx (D76435895)	2
SA	**MCSDSC-2.2 Relational database Management system.pdf** Document MCSDSC-2.2 Relational database Management system.pdf (D151484310)	5
W	URL: http://www.iete-elan.ac.in/SolQP/AC14.htm Fetched: 8/7/2021 5:47:19 AM	2
SA	**Sem III_ BCA_B21CA05DC.docx** Document Sem III_ BCA_B21CA05DC.docx (D165628090)	2
SA	**DBMS Block 2.pdf** Document DBMS Block 2.pdf (D149011992)	10
SA	**DBMS.docx** Document DBMS.docx (D142535242)	2
SA	**U234.pdf** Document U234.pdf (D109498494)	8
SA	**Database Management System Block 2.pdf** Document Database Management System Block 2.pdf (D164883642)	1
W	URL: https://slideplayer.com/slide/5143620/ Fetched: 4/3/2020 5:06:01 PM	3

Entire Document

CHAPTER 1 DATABASE SYSTEM
After reading this chapter, the reader will understand: ? The basic concept of
database and database management system ? The need of database system ? Limitations of traditional file processing system that led to the development of database management
system ? Advantages of database system ? Various developments in the field of database system ? Areas where the database system is used ? Cost and risk involved in database system ?
Different types of individuals that interact with the database system ? The concept of database schema and database instance ? Three-level DBMS architecture, which provides the abstract view of the database ?

73% **MATCHING BLOCK 1/319** SA 248E1130_RDBMS.docx (D165247738)

The concept of data independence, which helps the user to change the schema at one level of

the

SA

MATCHING BLOCK 2/319 248E1130_RDBMS.docx (D165247738)

database system without having to change the schema at the

other levels ? Database models, which provide the necessary means to achieve data abstraction Different types of DBMS languages ? Component modules of DBMS ? Difference between centralized and client/server database system ? Various types of database management systems ? The steps involved in designing the database system Storing data and retrieving information has been a necessity in all ages of business. The term data can be defined as a set of isolated and unrelated raw facts with an implicit meaning. Data can be anything such as, name of a person, a number, images, sound, etc. For example, 'Monica,' '25,' 'student,' etc., is a data. When the data is processed and converted into a meaningful and useful form, it is known as information. For example, 'Monica is 25 years old and she is a student.' is an information. For a business to be successful, a fast access to information is vital as important decisions are based on the information available at any point of time. Traditionally, the data was stored in voluminous repositories such as files, books, and ledgers. However, storing data and retrieving information from these repositories was a time-consuming task. With the development of computers, the problem of information storage and retrieval was resolved. Computers replaced tons of paper, file folders, and ledgers as the principal media for storing important information. Since then numerous techniques and devices have been developed to manage and organize the data so that useful information can be accessed easily and efficiently. The eternal quest for data management has led to the development of the database technology. A database can be defined as a collection of related data from which users can efficiently retrieve the desired information. A database can be anything from a simple collection of roll numbers, names, addresses, and phone numbers of students to a complex collection of sound, images, and even video or film clippings. Though databases are generally computerized, instances of non-computerized databases from everyday life can be cited in abundance. A dictionary, a phone book, a collection of recipes, and a TV guide are all common examples of non- computerized databases. The examples of computerized databases include customer files, employee rosters, books catalogue, equipment inventories, and sales transactions. In addition to the storage and retrieval of data, certain other operations can also be performed on a database. These operations include adding, updating, and deleting data. All these operations on a database are performed using a

66% **MATCHING BLOCK 3/319** SA BOOK SIZE.docx (D50092775)

database management system. A Database Management System (DBMS) is an integrated set of programs used to create and maintain a database.

The main objective

67% **MATCHING BLOCK 4/319** SA BOOK SIZE.docx (D50092775)

of a DBMS is to provide a convenient and effective method of defining, storing, retrieving, and manipulating the data contained in the database.

In addition, the DBMS must ensure the security of the database from unauthorized access and recovery of the data during system failures. It must also provide techniques for data sharing among several users. The database and the DBMS software are collectively known as database system. 5

1.1 NEED OF DATABASE SYSTEM The need of database systems arose in the early 1960s in response to the traditional file processing system. In the file processing system, the data is stored in the form of files, and a number of application programs are written by programmers to add, modify, delete, and retrieve data to and from appropriate files. New application programs are written as and when needed by the organization. For example, consider a bookstore that uses a file processing system to keep track of all the available books. The system maintains a file named BOOK to store information related to books. This information include book title, ISBN, price (in dollars), year of publishing, copyright date, category (such as language books, novels, and textbooks), number of pages, name of the author, name and address of the publisher, etc. In addition, the system has many application programs that allow users to manipulate the information stored in BOOK file. For example, the system may contain programs to add information about a new book, modify any existing book information, print the details of books according to their categories, etc. Learn More The ISBN is a unique commercial book identifier barcode. It is assigned to each edition of a book. An ISBN consists of several parts such as publisher code, title code, etc. These parts are usually separated by hyphens. Now, suppose a need arises to keep additional information about the publishers of the books that includes phone number and email id. To fulfil this need, the system creates a new file say PUBLISHER, which includes name, address, phone number, and email id of the publishers. New application programs are written and added to the system to manipulate the information in the PUBLISHER file. In this way, as the time goes by, more files and application programs are added to the system. This system of storing data in the form of files is supported by a conventional operating system. However, the file processing system has a number of disadvantages, which are given here. ?

Same information may be duplicated in several files. For example, the name and

the address of publisher are stored in BOOK file as well as in PUBLISHER file. This duplication of data is known as data redundancy, which leads to wastage of storage space. The other problem with file processing system is that the data may not be updated consistently. Suppose a publisher requests for a change in his address. Since the address of the publisher is stored in the BOOK as well as PUBLISHER file, both the files must be updated. If the address of the publisher is not modified in any of the two files, then the same publisher will have different address in two different files. This is known as data inconsistency. ? In any application, there are certain data integrity rules that need to be maintained. These rules could be in the form of certain conditions or constraints. In a file processing system, all these rules need to be explicitly programmed in all application programs, which are using that particular data item. For example, the integrity rule that each book should have a book title has to be implemented in all the application programs separately that are using the BOOK file. In addition, when new constraints are to be enforced, all the application programs should be changed accordingly. ? The file processing system lacks the insulation between program and data. This is because the file structure is embedded in the application program itself, thus, it is difficult to change the structure of a file as it requires changing all the application programs accessing it. For example, an application program in C++ defines the file structure using the keyword struct or class. Suppose the data type of the field ISBN is changed from string to number, changes have to be made in all the application programs that are accessing the BOOK file. ? Handling new queries is difficult, since it requires change in the existing application programs or requires a new application program. For example, suppose a need arises to print details of all the textbooks published by a particular publisher. One way to handle this request is to use the existing application program that prints details of the books according to their categories, and then manually generate the list of textbooks published by a particular publisher. Obviously, it is unacceptable. Alternatively, the programmer is requested to write a new application program. Suppose such a new application program is written. Now suppose after some time, a request 6 arises to filter the list of textbooks with price greater than $50. Then again we have to write a new application program. ? Since many users are involved in creating files and writing application programs, the various files in the system may have different file structures. Moreover, the programmers may choose different programming languages to write application programs. ? Application programs are added in an unplanned manner, due to which details of each file are easily available to every user. Thus, the file processing system lacks security feature. NOTE In spite of its limitations, the file processing system is still in use mainly for database backup. 1.2 ADVANTAGES OF DATABASE SYSTEM Database approach came into existence due to the bottlenecks of file processing system. In the database approach, the data is stored at a central location and is shared among multiple users. Thus, the main advantage of DBMS is centralized data management. The centralized nature of database system provides several advantages, which overcome the limitations of the conventional file processing system. These advantages are listed here. ? Controlled data redundancy: During database design, various files are integrated and each logical data item is stored at central location. This eliminates replicating the data item in different files, and ensures consistency and saves the storage space. Note that the redundancy in the database systems cannot be eliminated completely as there could be some performance and technical reasons for having some amount of redundancy. However, the DBMS should be capable of controlling this redundancy in order to avoid data inconsistencies. ? Enforcing data integrity: In database approach, enforcing data integrity is much easier. Various integrity constraints are identified by database designer during database design. Some of these data integrity constraints can be enforced automatically by the DBMS, and others may have to be checked by the application programs. ? Data sharing: The data stored in the database can be shared among multiple users or application programs. Moreover, new applications can be developed to use the same stored data. Due to shared data, it is possible to satisfy the data requirements of the new applications without having to create any additional data or with minimal modification. ? Ease of application development: The application programmer needs to develop the application programs according to the users' needs. The other issues like concurrent access, security, data integrity, etc., are handled by the DBMS itself. This makes the application development an easier task. ? Data security: Since the data is stored centrally, enforcing security constraints is much easier. The DBMS ensures that the only means of access to the database is through an authorized channel. Hence, data security checks can be carried out whenever access is attempted to sensitive data. To ensure security, a DBMS provides security tools such as user codes and passwords. Different checks can be established for each type of access (addition, modification, deletion, etc.) to each piece of information in the database. ? Multiple user interfaces: In order to meet the needs of various users having different technical knowledge, DBMS provides different types of interfaces such as query languages, application program interfaces, and graphical user interfaces (GUI) that include forms-style and menu-driven interfaces. A form-style interface displays a form to each user and user interacts using these forms. In menu-driven interface, the user interaction is through lists of options known as menus. ? Backup and recovery: The DBMS provides backup and recovery subsystem that is responsible for recovery from hardware and software failures. For example, if the failure occurs in between the transaction, the DBMS recovery subsystem either reverts back

the database to the state which existed prior to the start of the

transaction or resumes the transaction from the point it was interrupted so that its complete effect can be recorded in the database. In addition to centralized data management, DBMS also has some other advantages, which are discussed here. ? Program-data independence: The independence between the programs and the data is known as program-data independence (or simply data independence). It is an important characteristic of DBMS as it allows changing the 7 structure of the database without making any changes in the application programs that are using the database. To provide a high degree of data independence, the definition or the description of the database structure (structure of each file, the type and storage format of each data item) and various constraints on the data are stored separately in DBMS catalog. The information contained in the catalog is called the metadata (data about data). This independence is provided by a three-level DBMS architecture, which is discussed in Section 1.7. ? Data abstraction: The property of DBMS that allows program-data independence is known as data abstraction. Data abstraction allows the database system to provide an abstract view of the data to its users without giving the physical storage and implementation details. ? Supports multiple views of the data: A database can be accessed by many users and each of them may have a different perspective or view of the data. A database system provides a facility to define different views of the data for different users. A view is a subset of the database that contains virtual data derived from the database files but it does not exist in physical form. That is, no physical file is created for storing the data values of the view; rather, only the definition of the view is stored. Due to these advantages, the traditional file processing system for the bookstore is replaced by an Online Book database to maintain the information about various books, publishers, and authors. The author details include the name, address, URL, and phone number of author. Note that some of the authors also review the books and give ratings and feedbacks about the books. The database also maintains the details about the ratings given to the books by different reviewers. 1.3 DEVELOPMENTS IN DATABASE SYSTEM The database systems are divided into three generations.

41%	**MATCHING BLOCK 6/319**	SA	Advanced DBMS.pdf (D166063498)

The first generation includes the database systems based on hierarchical and network data model. The second generation includes the database systems based on relational data model, and the

third generation includes the database systems based on object-oriented and object-relational data model. In the early 1960s, the first general purpose DBMS, called the Integrated Data Store was designed by Charles Bachman. The development of magnetic tapes led to the automation of various data processing tasks such as payroll and inventory. In the late 1960s, the Information Management System (IMS) DBMS was developed by IBM. Two data models, network data model and hierarchical data model were also developed. During this period, use of hard disks instead of magnetic tapes facilitated direct access to data. In 1970, Edger Frank 'Ted' Codd (E.F. Codd) proposed the relational data model and non-procedural way to query data that led to the development of relational databases. Codd was felicitated with

100%	**MATCHING BLOCK 8/319**	SA	248E1130_RDBMS.docx (D165247738)

the prestigious Association of Computing Machinery Turing Award for his work.

In the 1980s, relational model became the dominant DBMS paradigm. SQL was standardized and the current standard known as SQL:1999 was adopted by ANSI/ISO. Research on parallel and distributed databases was carried out. During this period, several commercial relational databases such as IBM's DB2, Oracle, Informix UDS extended their systems to store new and complex data types such as images and text. In the early 1990s, SQL was designed mainly for decision support applications. In this period, vendors introduced products for parallel

54%	**MATCHING BLOCK 9/319**	SA	248E1130_RDBMS.docx (D165247738)

database. They also began to add object-relational support to their databases. In the late 1990s, with the advent of World Wide Web,

the use of DBMS to store data accessed through a Web browser became widespread. Database systems were designed to support very high transaction processing rates and 24 × 7 availability. Moreover, the database systems became more reliable during this period. In the early 2000s, XML databases and associated query language Xquery were developed to handle large amount of data and high transaction rates. The emergence of several enterprise resource planning (ERP) and management resource planning (MRP) packages like PeopleSoft, SAP, Siebel, etc., has added a substantial layer of application-oriented features on top of DBMS. These packages provide a general application layer to carry out a common set of tasks such as financial analysis, inventory control, human resource planning, etc. Different companies can customize this layer according to their need, which leads to the reduction in the overall cost of building the application layer. 1.4 APPLICATION AREAS OF DATABASE SYSTEM Database systems are widely used in different areas because of their numerous advantages. Some of the most common database applications are listed here. 8

? Airlines and railways: Airlines and railways use online databases for reservation, and for displaying the schedule information. ? Banking: Banks use databases for customer inquiry, accounts, loans, and other transactions. ? Education: Schools and colleges use databases for course registration, result, and other information. ? Telecommunications: Telecommunication departments use databases to store information about the communication network, telephone numbers, record of calls, for generating monthly bills, etc. ? Credit card transactions: Databases are used for keeping track of purchases on credit cards in order to generate monthly statements. ? E-commerce: Integration of heterogeneous information sources (for example, catalogs) for business activity such as online shopping, booking of holiday package, consulting a doctor, etc. ? Health care information systems and electronic patient record: Databases are used for maintaining the patient health care details. ? Digital libraries and digital publishing: Databases are used for management and delivery of large bodies of textual and multimedia data. ? Finance: Databases are used for storing information such as sales, purchases of stocks and bonds or data useful for online trading. ? Sales: Databases are used to store product, customer and transaction details. ? Human resources: Organizations use databases for storing information about their employees, salaries, benefits, taxes, and for generating salary checks. 1.5 COST AND RISK OF DATABASE SYSTEM Database systems have many advantages; however, before implementing database systems over the traditional file processing system, the cost and risk factors of implementing database systems should also be considered. The various cost and risk factors are given here. ? High cost: Installing new database system may require investment in hardware and software. The DBMS requires more main memory and disk storage. Moreover, DBMS is quite expensive. Therefore, a company needs to consider the overhead cost of implementing a new database system. ? Training new personnel: When an organization plans to adopt a database system, it may need to recruit or hire a specialized data administration group, which can coordinate with different user groups for designing views, establishing recovery procedures, fine tuning data structures to meet the organization requirements. Hiring such professionals is expensive. ? Explicit backup and recovery: A shared corporate database must be accurate and available at all times. Therefore, a system using online updation requires explicit backup and recovery procedures. ? System failure: When a computer system containing the database fails, all users have to wait until the system is functional again. Moreover, if DBMS or application program fails a permanent damage may occur to the database. NOTE In spite of cost and risk involved, database systems are widely used. 1.6 PEOPLE WHO INTERACT WITH DATABASES Database is an important asset of business. To design, use, and maintain such important asset many people are involved. The people who work with databases include database users, system analysts, application programmers, and database administrator (DBA). Database users are those who interact with the database in order to query and update the database, and generate reports. Database users are further classified into the following categories: ? Naive users: The users who query and update the database by invoking some already written application programs. For example, the owner of the bookstore enters the details of various books in the database by invoking appropriate application program. The na i ve user interacts with the database using form interface. 9
? Sophisticated users: The users, such as business analyst, scientist, etc., who are familiar with the facilities provided by a DBMS interact with the system without writing any application programs. Such users use database query language to retrieve information from the database to meet their complicated requirements. Things to Remember Though DBMS provides several facilities to access a database, but all kind of users need not learn about all these facilities. Naive users need to learn only some simple facilities of DBMS, whereas sophisticated users have to know most of the DBMS facilities in order to fulfill their complex requirements. ? Specialized users: The users who write specialized database programs, which are different from traditional data processing applications, such as banking and payroll management which use simple data types. Specialized users write applications such as

87%	MATCHING BLOCK 10/319	SA	248E1130_RDBMS.docx (D165247738)
computer-aided design systems, knowledge-base and expert systems that store data having complex data types.			

System analysts determine the requirements of the database users (especially na i ve users) to create a solution for their business need, and focus on non-technical and technical aspects. The nontechnical aspects involve defining system requirements, facilitating interaction between business users and technical staff, etc. Technical aspects involve developing the specification for user interface (application programs). Application programmers are the computer professionals who implement the specifications given by the system analysts, and develop application programs. They can choose tools, such as rapid application development (RAD) to develop the application program with minimal effort. The database application programmer develops application program to facilitate easy data access for the database users.

100%	MATCHING BLOCK 12/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

Database administrator (DBA) is a person who has central control over

both data and application programs. The responsibilities of DBA vary depending upon the job description and corporate and organization policies. Some of the responsibilities of DBA are given here. ? Schema definition and modification: The overall structure of the database is known as database schema. It is the responsibility of the DBA to create

92%	MATCHING BLOCK 11/319	W

the database schema by executing a set of data definition statements in DDL.

The DBA also carries out the changes to the schema according to the changing needs of the organization. ? New software installation: It is the responsibility of the DBA to install new DBMS software, application software, and other related software. After installation, the DBA must test the new software. ? Security enforcement and administration: DBA is responsible for establishing and monitoring the security of the database system. It involves adding and removing users, auditing, and checking for security problems. ? Data analysis: DBA is responsible for analyzing the data stored in the database, and studying its performance and efficiency in order to effectively use indexes, parallel query execution, etc. ? Preliminary database design: The DBA works along with the development team during the database design stage due to which many potential problems that can arise later (after installation) can be avoided. ? Physical organization modification: The DBA is responsible for carrying out the modifications in the physical organization of the database for better performance. ? Routine maintenance checks: The DBA is responsible for taking the database backup periodically in order to recover from any hardware or software failure (if occurs). Other routine maintenance checks that are carried out by the DBA are checking data storage and ensuring the availability of free disk space for normal operations, upgrading disk space as and when required, etc. 1.7 DBMS ARCHITECTURE AND DATA INDEPENDENCE The DBMS architecture describes how data in the database is viewed by the users. It is not concerned with how the data is handled and processed by the DBMS. The database users are provided

66%	MATCHING BLOCK 13/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

with an abstract view of the data by hiding certain details of how data is

physically stored. This enables the users to manipulate the

100%	MATCHING BLOCK 14/319	SA	Database System.pdf (D165747767)

data without worrying about where it is located or how it is actually stored.

In this architecture, the overall database description can be defined at three levels, namely, internal, conceptual, and external levels and thus, named three-level DBMS architecture. This architecture is proposed by

100%	MATCHING BLOCK 15/319	SA	BOOK SIZE.docx (D50092775)

ANSI/SPARC (American 10 National Standards Institute/Standards Planning and Requirements Committee)

and hence, is also known as ANSI/SPARC architecture. The three levels are discussed here. ? Internal level: It is the lowest level of data abstraction that deals with the physical representation of the database on the computer and thus, is also known as physical level. It describes how the data is physically stored and organized on the storage medium. At this level, various aspects are considered to achieve optimal runtime performance and storage space utilization. These aspects include storage space allocation techniques for data and indexes, access paths such as indexes, data compression and encryption techniques, and record placement. ? Conceptual level: This level of abstraction deals with the logical structure of the entire database and thus, is also known as

65%	MATCHING BLOCK 16/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

logical level. It describes what data is stored in the database, the relationships among the data

and complete view of the user's requirements without any concern for the physical implementation. That is, it hides the complexity of physical storage structures. The conceptual view is the overall view of the database and it includes all the information that is going to be represented in the database. ? External level: It is the highest level of abstraction that deals with the user's view of the database and thus, is also known as view level. In general, most of the users and application programs do not require the entire data stored in the database. The external level describes a

100%	MATCHING BLOCK 17/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

part of the database for a particular group of users.

It permits users to access data in a way that is customized according to their needs, so that the same data can be seen by different users in different ways, at the same time. In this way, it provides a powerful and flexible security mechanism by hiding the parts of the database from certain users, as the user is not aware of existence of any attributes that are missing from the view. These three levels are used to describe the schema of the database at various levels. Thus, the three-level architecture is also known as

41%	MATCHING BLOCK 18/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

three-schema architecture. The internal level has an internal schema, which describes the physical storage structure of the database. The conceptual level has a conceptual schema, which describes the structure of entire database. The external level has external schemas or user views, which describe the part of the database

according to a particular user's requirements, and hide the rest of the database from that user. The physical level is managed by the operating system under the direction of DBMS. The three-schema architecture is shown in Figure 1.1. Fig. 1.1 Three-schema architecture To understand the three-schema architecture, consider the three levels of the BOOK file in Online Book database as shown in Figure 1.2. In this figure, two views (view 1 and view 2) of the BOOK file have been defined at the external level. Different database users can see these views. The details of the data types are hidden from the users. At the conceptual level, the BOOK records are described by a type definition. The application programmers and the DBA generally work at this level of abstraction. At the internal level, the BOOK records are described as a block of consecutive storage locations such as words or bytes. The database users and the application programmers are not aware of these details; however, the DBA

100%	MATCHING BLOCK 19/319	SA	248E1130_RDBMS.docx (D165247738)

may be aware of certain details of the physical organization of the data. 11

Fig. 1.2 Three levels of Online Book database (BOOK file) In

100%	MATCHING BLOCK 20/319	SA	47F417BA15858476660.pdf (D123781188)

three-schema architecture, each user group refers only to its own external

view. Whenever a user specifies a request to generate a new external view, the DBMS must transform the request specified at external level into a request at conceptual level, and then into a request at physical level. If the user requests for data

39%	MATCHING BLOCK 21/319	SA	248E1130_RDBMS.docx (D165247738)

retrieval, the data extracted from the database must be presented according to the need of the user. This process of transforming the requests and results between various levels of DBMS architecture is known as mapping. The main advantage of three-schema architecture is that it provides data independence. Data independence is the ability

87%	MATCHING BLOCK 22/319	SA	BOOK SIZE.docx (D50092775)

to change the schema at one level of the database system without

having to change

75%	MATCHING BLOCK 23/319	SA	DBMS.docx (D68067493)

the schema at the other levels. Data independence is of two types, namely, logical data independence and physical data independence. ? Logical data independence: It is the ability to change the conceptual schema without affecting the external schemas

or application programs.
The conceptual schema may be changed due to change in constraints or addition of new data item or removal of existing data item, etc., from the database. The separation of the external level from the conceptual level enables the users to make changes at the conceptual level without affecting the external level or the application programs. For example, if a new data item, say Edition is added to the BOOK file, the two views (view 1 and view 2 shown in Figure 1.2) are not affected. ? Physical data independence: It

60%	MATCHING BLOCK 24/319	SA	47F417BA15858476660.pdf (D123781188)

is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be

changed due to several reasons such as for creating additional access structure, changing the storage structure, etc. The separation of internal schema from the conceptual schema facilitates physical data independence. Logical data independence is more difficult to achieve than the physical data independence because the application programs are always dependent on the logical structure of the database. Therefore, the change in the logical structure of the database may require change in the application programs. 1.8 DATABASE MODELS As discussed earlier, the main objective of database system is to highlight only the essential features and to hide the storage and data organization details from the user. This is known as data abstraction. A database model provides the necessary means to achieve data abstraction. A database model or simply a data model is an abstract model that describes how the data is represented and used. A data model consists of a set of data structures and conceptual tools that is used to describe the structure (data types, relationships, and constraints) of a database. A data model not only describes the structure of the data, it also defines a set of operations that can be performed on the data. A data model generally consists of data model theory, which is a formal description of how data may be structured and used, and data model instance, which is a practical data model designed for a particular application. The process of applying a data model theory to create a data model instance is known as data modeling. 12

Depending on the concept they use to model the structure of the database, the data models are categorized into three types, namely, high-level or conceptual data models, representational or implementation data models and low-level or physical data models. 1.8.1 Conceptual Data Model Conceptual data model describes the information used by an organization in a way that is independent of any implementation-level issues and details. The main advantage of conceptual data model is that it is independent of implementation details and hence, can be understood even by the end users having non-technical background. The most popular conceptual data model, that is, entity-relationship (E-R) model is discussed in detail in Chapter 02. 1.8.2 Representational Data Model The representational or implementation data models hide some data storage details from the users; however, can be implemented directly on a computer system. Representational data models are used most frequently in all traditional commercial DBMSs. The various representational data models are discussed here. Hierarchical Data Model The hierarchical data model is the oldest type of data model, developed by IBM in 1968. This data model organizes the data in a tree-like structure, in which each child node (also known as dependents) can have only one parent node. The database based on the hierarchical data model comprises a set of records connected to one another through links. The link is an association between two or more records. The top of the tree structure consists of a single node that does not have any parent and is called the root node. The root may have any number of dependents; each of these dependents may have any number of lower level dependents. Each child node can have only one parent node and a parent node can have any number of (many) child nodes. It, therefore, represents only one-to-one and one-to-many relationships. The collection of same type of records is known as a record type. Figure 1.3 shows the hierarchical model of Online Book database. It consists of three record types, namely, PUBLISHER, BOOK, and REVIEW. For simplicity, only few fields of each record type are shown. One complete record of each record type represents a node. The main advantage of the hierarchical data model is that the data access is quite predictable in the structure and, therefore, both the retrieval and updates can be highly optimized by the DBMS. Fig. 1.3 Hierarchical data model for Online Book database However, the main drawback of this model is that the links are 'hard coded' into the data structure, that is, the link is permanently established and cannot be modified. The hard coding makes the hierarchical model rigid. In addition, the physical links make it difficult to expand or modify the database and the changes require substantial redesigning efforts. Network Data Model 13
The first specification of network data model was presented by Conference on Data Systems Languages (CODASYL) in 1969, followed by the second specification in 1971. It is powerful but complicated. In a network model the data is also represented by a collection of records, and relationships among data are represented by links. However, the link in a network data model represents an association between precisely two records. Like hierarchical data model, each record of a particular record type represents a node. However, unlike hierarchical data model, all the nodes are linked to each other without any hierarchy. The main difference between hierarchical and network data model is that in hierarchical data model, the data is organized in the form of trees and in network data model, the data is organized in the form of graphs. The main advantage of network data model is that a parent node can have many child nodes and a child can also have many parent nodes. Thus,

85%	**MATCHING BLOCK 25/319**	SA	all in one.pdf (D109420358)

the network model permits the modeling of many-to-many relationships in data. The

main limitation of the network data model is that it can be quite complicated to maintain all the links and a single broken link can lead to problems in the database. In addition, since there are no restrictions on the number of relationships, the database design can become complex. Figure 1.4 shows the network model of Online Book database. Fig. 1.4 Network data model for Online Book database NOTE Links in hierarchical and network data models are generally implemented through pointers. Relational Data Model The relational data model was developed by E. F. Codd in 1970. In the relational data model, unlike the hierarchical and network models, there are no physical links. All data is maintained in the form of tables (generally, known as relations) consisting of rows and columns. Each row (record) represents an entity and a column (field) represents an attribute of the entity. The relationship between the two tables is implemented through a common attribute in the tables and not by physical links or pointers. This makes the querying much easier in a relational database system than in the hierarchical or network database systems. Thus, the relational model has become more programmer friendly and much more dominant and popular in both industrial and academic scenarios. Oracle, Sybase, DB2, Ingres, Informix, MS-SQL Server are few of the popular relational DBMSs. Figure 1.5 shows the relational model of Online Book database. 14
Fig. 1.5 Relational data model for Online Book database NOTE The network, hierarchical, and relational data models are known as record-based data models as these models represent data in the form of records. Object-Based Data Model In the recent years, the object-oriented paradigm has been applied to database technology, creating two new data models known as

object-oriented data model and object-relational data model. The object-oriented data model

extends the concepts of object-oriented programming language with persistence, versioning, concurrency control, data recovery, security, and other database capabilities. On the other hand, the object-relational data model is an extension of relational data model. It combines the features of both the relational data model and object-oriented data model. Semistructured Data Model Unlike other

data models, where every data item of a particular type must have the same set of attributes, the

semistructured data model allows

individual data items of the same type to have different set of attributes.

In semistructured data model, the information about the description of the data (schema) is contained within the data itself, which is sometimes called self-describing data. In such databases there is no clear separation between the data and the schema and thus, allowing data of any type. Semistructured data model has recently emerged as an important topic of study for different reasons given here. ? There are data sources such as the Web, which is to be treated as databases; however, they cannot be constrained by a schema. ? The need of flexible format for data exchange between heterogeneous databases. ? To facilitate browsing of data. Semistructured data model facilitates data exchange among heterogeneous data sources. It helps to discover new data easily and store it. It also facilitates querying the database without knowing the data types. However, it loses the data type information. Learn More XML (Extensible Markup Language) is a means of representing semistructured data. XML enables to define schema using XML schema language, which allows varying levels of schema over data elements. 15
Fig. 1.6 Database schema for Online Book database 1.8.3 Physical Data Model Physical data model describes the data in terms of a collection of files, indices, and other storage structures such as record formats, record ordering, and access paths. This model specifies how the database will be executed in a particular DBMS software such as Oracle, Sybase, etc., by taking into account the facilities and constraints of a given database management system. It also describes how the data is stored on disk and what access methods are available to it. NOTE An access path (for example, an index) is a structure that helps in efficient searching and retrieval of data from a particular database. 1.9 DATABASE SCHEMA VERSUS DATABASE INSTANCE While working with any data model, it is necessary to distinguish between the overall design or description of the database (database schema) and the database itself. As discussed earlier, the overall design or description of the database is known as database schema or simply schema. The database schema is also known as intension of the database, and is specified while designing the database. Figure 1.6 shows the database schema for Online Book database. The data in the database at a particular point of time is known as database instance or database state or snapshot. The database state is also called an extension of the schema. The various states of the database are given here. ? Empty state: When a new database is defined, only its schema is specified. At this point, the database is said to be in empty state as it contains no data. ? Initial state: When the database is loaded with data for the first time, it is said to be in initial state. ? Current state: The data in the database is updated frequently. Thus, at any point of time, the database is said to be in the current state. The DBMS is responsible to check whether the database state is valid state. Thus, each time the database is updated, DBMS ensures that the database remains in the valid state. The DBMS refers to DBMS catalog where the metadata is stored in order to check whether the database state satisfies the structure and constraints specified in the schema. Figure 1.7 shows an example of an instance for PUBLISHER schema. 16

Fig. 1.7 An example of instance - PUBLISHER instance The schema and instance can be compared with a program written in a programming language. The database schema is similar to a variable declared along with the type description in a program. The variable contains a value at a given point of time. This value of a variable corresponds to an instance of a database schema. NOTE The database schema changes rarely, whereas the data in the database may change frequently according to the requirements. The process of making any changes in the schema is known as schema evolution. 1.10 DBMS LANGUAGES The main objective of a database management system is to allow its users to perform a number of operations on the database such as insert, delete, and retrieve data in abstract terms without knowing about the physical representations of data. To provide the various facilities to different types of users, a DBMS normally provides one or more specialized programming languages called Database (or DBMS) Languages. Learn More In addition to DDL and DML, DBMS also provides two more languages, namely, data control language (DCL) and transaction control language (TCL). Data control language is used to create user roles, grant permissions, and control access to database by securing it. Transaction control language is used to manage different transactions occurring within a database. The DBMS mainly provides two database languages, namely, data definition language and data manipulation language to implement the databases. Data definition language (DDL) is used for defining the database schema. The DBMS comprises DDL compiler that identifies and stores the schema description in the DBMS catalog. Data manipulation language (DML) is used to manipulate the database. 1.10.1 Data Definition Language In DBMSs where no strict separation between the levels of the database is maintained, the data definition language is used to define the conceptual and internal schemas for the database. On the other hand, in DBMSs, where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. In such DBMSs, a separate language, namely, storage definition language (SDL) is used to define the internal schema. Some of the DBMSs that are based on true three-schema architecture use a third language, namely, view definition language (VDL) to define the external schema. 17
The DDL statements are also used to specify the integrity rules (constraints) in order to maintain the integrity of the database. The various integrity constraints are domain constraints, referential integrity, assertions and authorization. These constraints are discussed in detail in subsequent chapters. Like any other programming language, DDL also accepts input in the form of instructions (statements) and generates the description of schema as output. The output is placed in the data dictionary, which is a special type of table containing metadata. The DBMS refers the data dictionary before reading or modifying the data. Note that the database users cannot update the data dictionary; instead it is only modified by database system itself. 1.10.2 Data Manipulation Language Once the database schemas are defined and the initial data is loaded into the database, several operations such as retrieval, insertion, deletion, and modification can be applied to the database. The DBMS provides data manipulation language (DML) that enables users to retrieve and manipulate the data. The statement which is used to retrieve the information is called a query. The part of the DML used to retrieve the information is called a query language. However, query language and DML are used synonymously though technically incorrect. The DML are of two types, namely, non- procedural DML and procedural DML. The non-procedural or high-level or declarative DML enables to specify the complex database operations concisely. It requires a user to specify what data is required without specifying how to retrieve the required data. For example, SQL (Structured Query Language) is a non-procedural query language as it enables user to easily define the structure or modify the data in the database without specifying the details of how to manipulate the database. The high-level DML statements can either be entered interactively or embedded in a general purpose programming language. On the other hand, the procedural or low-level DML requires user to specify what data is required and how to access that data by providing step- by-step procedure. For example, relational algebra is procedural query language, which consists of set of operations such as select, project, union, etc., to manipulate the data in the database. Relational algebra and SQL are discussed in Chapters 04 and 05, respectively. 1.11 COMPONENT MODULES OF DBMS A database system being a complex software system is partitioned into several software components that handle various tasks such as data definition and manipulation, security and data integrity, data recovery and concurrency control, and performance optimization (see Figure 1.8). ? Data definition: The DBMS provides functions to define the structure of the data. These functions include defining and modifying the record structure, the data type of fields, and the various constraints to be satisfied by the data in each field. It is the responsibility of database administrator to define the database, and make changes to its definition (if required) using the DDL and other privileged commands. The DDL compiler component of DBMS processes these schema definitions, and stores the schema descriptions in the DBMS catalog (data dictionary). Other DBMS components then refer to the catalog information as and when required. ? Data manipulation: Once the data structure is defined, data needs to be manipulated. The manipulation of data includes insertion, deletion, and modification of records. The functions that perform these operations are also part of the DBMS. These functions can handle planned as well as unplanned data manipulation needs. o The queries that are defined as a part of the application programs are known as planned queries. The application programs are submitted to a precompiler, which extracts DML commands from the application program and send them to DML compiler for compilation.

50%	MATCHING BLOCK 32/319	SA	Database System.pdf (D165747767)

The rest of the program is sent to the host language compiler. The object codes of both the DML commands and the rest of the program are linked and sent to the query evaluation engine for execution. o The

sudden queries that are executed as and when the need arises are known as unplanned queries (interactive queries). These queries are compiled by the query complier, and then optimized by the query optimizer. The query optimizer consults the data dictionary for statistical and other physical information about the stored data (query optimization is discussed in detail in Chapter 08). The optimized query is finally passed to the query evaluation engine for execution. The na i ve users of the 18
database can also query and update the database by using some already given application program interfaces. The object code of these queries is also passed to query evaluation engine for processing. ? Data security and integrity: The DBMS contains functions, which handle the security and integrity of data stored in the database. Since these functions can be easily invoked by the application, the application programmer need not code these functions in the programs. ? Concurrency and data recovery: The DBMS also contains some functions that deal with the concurrent access of records by multiple users and the recovery of data after a system failure. The concurrency control techniques and database recovery is discussed in Chapters 10 and 11, respectively. ? Performance optimization: The DBMS has a set of functions that optimize the performance of the queries by evaluating the different execution plans of a query and choosing the best among them. In this way, the DBMS provides

100%	MATCHING BLOCK 29/319	W

an environment that is both convenient and efficient to use

when there is a large volume of data and many transactions need to be processed concurrently. Fig. 1.8
Component modules of DBMS 1.12 CENTRALIZED AND CLIENT/SERVER DATABASE SYSTEMS
In centralized database systems, the database system, application programs, and user-interface all are executed on a single system and dummy terminals are connected to it. The processing power of single system is utilized and dummy terminals are used only to display the information. As the personal computers became faster, more powerful, and cheaper, the database system started to exploit the available processing power of the system at the user’s side, which led to the development of client/server architecture. In client/server architecture, the processing power of the computer system at the user’s end is utilized by processing the user-interface on that system. A client is a computer system that sends request to the server connected to the network, and a server is a computer system that receives the request, processes it, and returns the requested information back to the client. Client and server are usually present at different sites. The end users (remote database users) work on client computer system and database system runs on the server. Servers can be of several types, for example, file servers, printer servers, web servers, database servers, etc. The client machines have user interfaces that help users to utilize the servers. It also provides users the local processing power to run local applications on the client side. 19
There are two approaches to implement client/server architecture. In the first approach, the user interface and application programs are placed on the client side and the database system on the server side. This architecture is called two-tier architecture. The application programs that reside at the client side invoke the DBMS at the server side. The application program interface standards like Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) are used for interaction between client and server. Figure 1.9 shows two-tier architecture. The second approach, that is, three-tier architecture is primarily used for web-based applications. It adds intermediate layer known as application server (or web server) between the client and the database server. The client communicates with the application server, which in turn communicates with the database server. The application server stores the business rules (procedures and constraints) used for accessing data from database server. It checks the client’s credentials before forwarding a request to database server. Hence, it improves database security. When a client requests for information, the application server accepts the request, processes it, and sends corresponding database commands to database server. The database server sends the result back to application server which is converted into GUI format and presented to the client. Figure 1.10 shows the three-tier architecture. Fig. 1.9 Two-tier architecture Fig. 1.10 Three-tier architecture 1.13 CLASSIFICATION OF DBMSS The DBMSs can be classified into different categories on the basis of several criteria such as the data model they are using, number of users they support, number of sites over which the database is distributed, and the purpose they serve. Based on Data Models 20

The various data models have already been discussed. Depending on the data model they use, the DBMSs can be classified as hierarchical, network, relational, object-oriented, and object-relational. Among these, the hierarchical and network data models are the older data models and now known as legacy data models. Some of the old applications still run on the database systems based on these models. Most of the popular and current commercial DBMSs are based on relational data model. The object-based data models have been implemented in some DBMSs; however, have not become popular. Due to the popularity of relational databases, the object-oriented concepts have been introduced in these databases that led to the development of a new class of DBMSs called object-relational DBMSs. Learn More MYSQL and PostgreSQL are open source (free) DBMS products that are supported by third-party vendors with additional services. Based on Number of Users Depending on the number of users the DBMS supports, it is divided into two categories, namely, single-user system and multi-user system. In single-user system the database resides on one computer and is only accessed by one user at a time. The user may design, maintain, and write programs for accessing and manipulating the database according to the requirements, as well as perform all the user roles. The user may also hire database system designers to design a system. In such a case, the single user performs the role of end user only. However, in most enterprises the large amount of data is to be managed and accessed by multiple users and thus, requires multi-user systems. In multi-user system, multiple users can access the database simultaneously. In multi-user DBMS, the data is both integrated and shared. For example, the Online Book database is a multi-user database system in which the data of books, authors, and publishers are stored centrally and can be accessed by many users. Based on Number of Sites Depending on

100%	MATCHING BLOCK 30/319	SA	Advanced DBMS.pdf (D166063498)

the number of sites over which the database is distributed,

it is divided into two types, namely, centralized and distributed database systems. Centralized database systems run on a single computer system. Both the database and DBMS software reside at a single computer site. The user interacts with the centralized system through a dummy terminal connected to it for information retrieval. In distributed database systems, the database and DBMS are distributed over several computers located at different sites. The computers

75%	MATCHING BLOCK 31/319	SA	Advanced DBMS.pdf (D166063498)

communicate with each other through various communication media such as high-speed networks or

telephone lines. Distributed databases can be classified as

93%	MATCHING BLOCK 35/319	SA	248E1130_RDBMS.docx (D165247738)

homogeneous and heterogeneous. In homogeneous distributed database system, all sites have identical database management system software,

whereas in heterogeneous distributed database system, different sites use different database management system software. Based on the Purpose Depending on the purpose the DBMS serves, it can be classified as general purpose or specific purpose. DBMS is a general purpose software system. It can; however, be designed for specific purposes such as airline or railway reservation. Such systems cannot be used for other applications without major changes. These database systems fall under the category of online transaction processing (OLTP) systems. Online transaction processing system is specifically used for data entry and retrieval. It supports large number of concurrent transactions without excessive delays. An automatic teller machine (ATM) for a bank is an example of online commercial transaction processing application. The OLTP technology is used in various industries, such as banking, airlines, supermarkets, manufacturing, etc. 1.14 DATABASE DESIGN PROCESS The overall database design process has to follow a series of steps. The systematic process of designing a database is known as design methodology. Database design involves understanding operational and business needs of an organization, modeling the specified requirements, and realizing the requirements using a database. The goal of designing a database is to produce efficient, high quality, and minimum cost database. In large organizations, database administrator 21

(DBA) is responsible for designing an efficient database system. He is responsible for controlling the database life-cycle process. The overall database design and implementation process consists of several phases. 1. Requirement collection and analysis: It is the process of knowing and analyzing the expectations of the users for the new database application in as much detail as possible. A team of analysts or requirement experts are responsible for carrying out the task of requirement analysis. They review the current file processing system or DBMS system, and interact with the users extensively to analyze the nature of business area to be supported and to justify the need for data and databases. The initial requirements may be informal, incomplete, inconsistent, and partially incorrect. The requirement specification techniques such as object-oriented analysis (OOA), data flow diagrams (DFDs), etc., are used to transform these requirements into better structured form. This phase can be quite time-consuming; however, it plays the most crucial and important role in the success of the database system. The result of this phase is the document containing the specification of user requirements. 2. Conceptual database design: In this phase, the database designer selects a suitable data model and translates the data requirements resulting from previous phase into a conceptual database schema by applying the concepts of chosen data model. The conceptual schema is independent of any specific DBMS. The main objective of conceptual schema is to provide a detailed overview of the organization. In this phase, a high-level description of the data and constraints are developed. The entity-relationship (E-R) diagram is generally used to represent the conceptual database design. The conceptual schema should be expressive, simple, understandable, minimal, and formal. 3. Choice of a DBMS: The choice of a DBMS depends on many factors such as cost, DBMS features and tools, underlying model, portability, and DBMS hardware requirements. The technical factors that affect the choice of a DBMS are the type of DBMS (relational, object, object-relational, etc.), storage structures and access paths that DBMS supports, the interfaces available, the types of high-level query languages, and the architecture it supports (client/server, parallel or distributed). The various types of costs that must be considered while choosing a DBMS are software and hardware acquisition cost, maintenance cost, database creation and conversion cost, personnel cost, training cost, and operating cost. 4. Logical database design: Once an appropriate DBMS is chosen, the next step is to map

the high-level conceptual schema onto the implementation data model of the

selected DBMS. In this phase, the database designer moves

from an abstract data model to the implementation of the database. In

case of relational model, this phase generally consists of mapping the E-R model into a relational schema. 5. Physical database design: In this phase, the physical features such as storage structures, file organization, and access paths for the database files are specified to achieve good performance. The various options for file organization and access paths include various types of indexing, clustering of records, hashing techniques, etc. 6. Database system implementation: Once the logical and physical database designs are completed, the database system can be implemented. DDL statements of the selected DBMS are used and compiled to create the database schema and database files, and finally the database is loaded with the data. 7. Testing and evaluation: In this phase, the database is tested and fine-tuned for the performance, integrity, concurrent access, and security constraints. This phase is carried out in parallel with application programming. If the testing fails, various actions are taken such as modification of physical design, modification of logical design or upgrade or change DBMS software or hardware. Note that once the application programs are developed, it is easier to change the physical database design. However, it is difficult to modify the logical database design as it may affect the queries (written using DML commands) embedded in the program code. Thus, it is necessary to carry out the design process effectively before developing the application programs. While designing a database schema, it is necessary to avoid two major issues, namely, redundancy and incompleteness. These problems may lead to bad database design. The anomalies (an inconsistent, incomplete or contradictory state of the database) that can arise due to these problems are discussed in detail in Chapter 06. NOTE The database design phases, namely, conceptual, logical, and physical are discussed in subsequent chapters. 22
SUMMARY 1. The term data can be defined as a set of isolated and unrelated raw facts with an implicit meaning. 2. When the data is processed and converted into a meaningful and useful form, it is known as information. 3. A database can be defined as a collection of related data from which users can efficiently retrieve the desired information. 4.

61%	MATCHING BLOCK 36/319	SA	BOOK SIZE.docx (D50092775)

A Database Management System (DBMS) is an integrated set of programs used to create and maintain a database.

The main objective

67%	MATCHING BLOCK 38/319	SA	BOOK SIZE.docx (D50092775)

of a DBMS is to provide a convenient and effective method of defining, storing, retrieving, and manipulating the data contained in the database. 5.

The database and the DBMS software are collectively known as database system. 6. Database approach came into existence due to the bottlenecks of file processing system. In the database approach, the data is stored at a central location and is shared among multiple users. Thus, the main advantage of DBMS is centralized data management. Other advantages of DBMS are program-data independence, data abstraction, and support for multiple views of data. 7. Centralized data management in DBMS offers many advantages such as controlled data redundancy, data consistency, enforcing data integrity, data sharing, ease of application development, data security, multiple user interfaces, and backup and recovery. 8. The database systems are divided into three generations.

41%	MATCHING BLOCK 37/319	SA	Advanced DBMS.pdf (D166063498)

The first generation includes the database systems based on hierarchical and network data model. The second generation includes the database systems based on relational data model and the

third generation includes the database systems based on object-oriented and object-relational data model. 9. The most common database application areas are airlines and railways, banking, education, telecommunications, credit card transactions, e-commerce, health care information systems and electronic patient records, digital libraries and digital publishing, finance, sales, and human resources. 10. The people who work with databases include database users, system analysts, application programmers, and

89%	MATCHING BLOCK 39/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

database administrator (DBA). Database administrator (DBA) is a person who has central control over

both data and application programs. 11. The DBMS architecture describes how data in the database is viewed by the users. It is not concerned with how the data is handled and processed by the DBMS. In this architecture, the overall database description can be defined at three levels, namely, internal, conceptual, and external levels and thus, named three-level DBMS architecture. This architecture is proposed by ANSI/SPARC, and hence is also known ANSI/SPARC architecture. 12. Internal level is the lowest level of data abstraction that deals with the physical representation of the database on the computer. Conceptual level deals with the logical structure of the entire database and thus, is also known as logical level. External level is the highest level of abstraction that deals with the user's view of the database and thus is also known as view level. 13. The process of transforming the requests and results between various levels of DBMS architecture is known as mapping. 14. The main advantage of three-schema architecture is that it provides data independence. Data independence is the ability

87%	MATCHING BLOCK 40/319	SA	BOOK SIZE.docx (D50092775)

to change the schema at one level of the database system without

having to change

72%	MATCHING BLOCK 41/319	SA	DBMS.docx (D68067493)

the schema at the other levels. Data independence is of two types, namely, logical data independence and physical data independence. 15.

The process of highlighting only the essential features of a database system and hiding the storage and data organization details from the user is known as data abstraction. 16. A database model or simply a data model is an abstract model that describes how the data is represented and used. It not only describes the structure of the data but also defines a set of operations that can be performed on the data. 17. Depending on the concept they use to model the structure of the database, the data models are categorized into three types, namely, high-level or conceptual data models, representational or implementation data models, and low-level or physical data models. 18. Conceptual data model describes the information used by an organization in a way that is independent of any implementation-level issues and details. 23
19. The representational or implementation data models hide some data storage details from the users; however, can be implemented directly on a computer system. The various representational data models are hierarchical, network, relational, object-based, and semistructured data model. 20. Physical data model describes the data in terms of a collection of files, indices, and other storage structures such as record formats, record ordering, and access paths. 21. The overall design or description of the database is known as database schema or simply schema. The database schema is also known as intension of the database, and is specified while designing the database. The data in the database at a particular point of time is known as database instance or database state or snapshot. The database state is also called an extension of the schema. 22. To provide various facilities to different types of users, a DBMS normally provides one or more specialized programming languages often called Database (or DBMS) languages. 23. The DBMS mainly provides two database languages, namely, data definition language and data manipulation language to implement the databases. 24. Data definition language (DDL) is used for defining the database schema. The DBMS comprises DDL compiler that identifies and stores the schema description in the DBMS catalog. Data manipulation language (DML) is used to manipulate the database. 25. A database system being a complex software system is partitioned into several software components that handle various tasks such as data definition and manipulation, security and data integrity, data recovery and concurrency control, and performance optimization. 26. In client/server architecture, the client is the computer system that sends request to the server connected to the network and the server is a computer system that receives the request, processes it, and returns the requested information to the client. 27. There are two approaches of implementing client/server architectures, namely, two-tier architecture and three- tier architecture. 28. The DBMS can be classified into different categories on the basis of several criteria, namely, data models, number of users, number of sites, and purpose. 24
CHAPTER 2 CONCEPTUAL MODELING After reading this chapter, the reader will understand: ? Basic concept of entity-relationship model ? Entities and their attributes ? Entity types and entity set ? Different types of attributes ? Key attribute, and the concept of superkey, candidate key and primary key ? Concept of weak and strong entity types ? Relationships among various entity types ? Constraints on relationship types ? Binary versus ternary relationships ? E-R diagram notations ? Features of enhanced E-R model that include specialization, generalization, and aggregation ? Constraints on specialization and generalization ? Alternative notations of E-R diagram ? Unified modeling language (UML), a standard language used to implement object data modeling ? Categories of UML diagrams that include structural diagrams and behavioural diagrams As discussed in the earlier chapter, the overall database design and implementation process starts with requirement collection and analysis and specifications. Once the user's requirements have been specified, the next step is to construct an abstract or conceptual model of a database based on those requirements. This phase is known as conceptual modeling or semantic modeling. Conceptual modeling is an important phase in designing a successful database. It represents various pieces of data and their relationships at a very high-level of abstraction. It mainly

focuses on what data is required and how it should be organized rather than what operations

are to be performed on the data. The conceptual model is a concise and permanent description of the database requirements that facilitates an easy communication between the users and the developers. It is simple enough to be understood by the end users; however, detailed enough to be used by a database designer to create the database. The model is independent of any hardware (physical storage) and software (DBMS) constraints and hence can be modified easily according to the changing needs of the user. The conceptual model can be represented using two major approaches, namely, entity-relationship modeling and the object modeling. This chapter discusses the entity-relationship (E-R) model and its extended features, which are used for designing a conceptual model of any database application. It also discusses the entity-relationship diagram, a basic component of the E-R model used to graphically represent the data objects. An E-R diagram helps in modeling the data representation component of a software system. The other components of the software system such as user interactions with the system, functional modules and their interactions, etc. are specified with the help of Unified Modeling Language (UML), which is a standard methodology used in object modeling. 2.1 CONCEPTS OF ENTITY-RELATIONSHIP MODEL The entity-relationship (E-R) model is the most popular conceptual model used for designing a database. It was originally proposed by Dr.

65%	MATCHING BLOCK 43/319	SA	47F417BA15858476660.pdf (D123781188)
Peter Chen in 1976 as a way to unify the network and relational database views. The E-R model views the real world as			

a set of basic objects (
known as entities), their characteristics (known as attributes), and associations among these objects (known as relationships). The entities, attributes, and relationships are the basic constructs of an E-R model. The information gathered from the user forms the basis for designing the E-R model. The nouns in the requirements specification document are represented as the entities, the additional nouns that describe the nouns corresponding to entities form the attributes, and the verbs become the relationships among various entities. The E-R model has several advantages listed as follows. 25
Things to Remember The E-R model does not actually gives us a database description; rather it gives us an intermediate step from which we can easily create a database. ? It is simple and easy to understand and thus, can be used as an effective communication tool between database designers and end users. ? It captures the real-world data requirements in a simple, meaningful, and logical way. ? It can be easily mapped onto the relational model. The basic constructs, that is, the entities and attributes of the E-R model can be easily transformed into relations (or tables) and columns (or fields) in a relational model. ? It can be used as a design plan and can be implemented in any database management software. 2.1.1 Entity and its Attributes An entity is any distinguishable object that has an independent existence in the real world. It includes all those 'things' of an organization about which the data is collected. For example, each book, publisher, and author in the Online Book database (discussed in Chapter 01) is an entity. An entity can exist either physically or conceptually. If an entity has a physical existence, it is termed as tangible or concrete entity (for example, a book, an employee, a place or a part) and if it has a conceptual existence, it is termed as non-tangible or abstract entity (for example, an event, a job title or a customer account). Each entity is represented by a set of attributes. The attributes are the properties of an entity that characterize and describe it. Figure 2.1 shows the BOOK entity b 1 described by the attributes Book_title. Price (in dollars), ISBN, Year, Page_count and Category. Similarly, the PUBLISHER entity p 1 is described by the attributes P_ID, Name, Address, Phone and Email_ID. Here, b 1 and p 1 are the instances of the entity types. The entity types and instances are discussed later in this section. Each attribute has a particular value. For example, the attributes Book_title, Price, ISBN, P_ID, Year, Page_count and Category can have the values Ransack, 22, 001-354-921-1, 2006, 200 and Novel, respectively, (as shown in Figure 2.1). Similarly, the attributes P_ID, Name, Address, Phone and Email_ID can have the values P001, Hills Publications, 12 Park Street Atlanta Georgia 30001, 7134019, h_pub@hills.com, respectively. Each attribute can accept a value from a set of permitted values, which is called the domain or the value set of the attribute. For example, the domain of the attribute Page_count is a set of positive integers. On the other hand, the category of a book can be textbook, language book or novel. Thus, the domain of the attribute Category is {Textbook, Language book, Novel}. Fig. 2.1 Entity instances b 1 (BOOK) and p 1 (PUBLISHER), and their attributes Entity Types and Entity Sets A set or a collection of entities that share the same attributes but different values, is known as an entity type. For example, the set of all publishers who publish books can be defined as the entity type PUBLISHER. Similarly, a set of books written by the authors can be defined as the entity type BOOK. A specific occurrence of an entity type is called its instance. For 26

example, the novel Ransack and the publisher Hills Publications are the instances of the entity types BOOK and PUBLISHER, respectively. The collection of all instances of a particular entity type in the database at any point of time is known as an entity set. Each entity set is referred to by its name and attribute values. Note that both the entity set and the entity type are usually referred to by the same name. For example, BOOK refers to both the type of entity and the current set of all book entities. An entity type describes the schema (intension) for the entity set that shares the same structure. The entity set, on the other hand, is called the extension of the entity type. Figure 2.2 shows two entity types, BOOK and PUBLISHER, and a list of attributes for each of them. Some individual instances of each entity type are also shown along with their attribute values. Fig. 2.2 Entity types and entity sets Types of Attributes The attributes of an entity are classified into the following categories: ? Identifying and descriptive attributes: The attribute that is used to uniquely identify an instance of an entity is known as identifying attribute or simply an identifier. For example, the attribute P_ID of the entity type PUBLISHER is identifying attribute, as two publishers cannot have the same publisher ID. Similarly, the attribute ISBN of the entity type BOOK is also an identifier, as two different books cannot have the same ISBN. A descriptive attribute or simply a descriptor, on the other hand, describes a non-unique characteristic of an entity instance. For example, the attributes Price and Page_count are the descriptive attributes as two books can have the same price and number of pages. ? Simple and composite attributes: The attributes that are indivisible are known as simple (or atomic) attributes. For example, the attributes Book_title, Price, Year, Page_count, and Category of the entity type BOOK are simple attributes as they cannot be further divided into smaller subparts. On the other hand, composite attributes can be divided into smaller subparts. For example, the attribute Address can be further divided into House_number, Street, City, State and Zip_code (as shown in Figure 2.3). Composite attributes help in grouping the related attributes together. They are useful in those situations where the user sometimes wants to refer to an entire attribute and sometimes to only a component of the attribute. In case the user does not want to refer to the individual components, there is no need to subdivide the composite attribute into its component attributes. 27
Fig. 2.3 Composite attribute ? Stored

84%	MATCHING BLOCK 44/319	W
and derived attributes: In some cases, two or more attribute values are related		

in such a way that the value of one attribute can be determined from the value of other attributes or related entities. For example, if the entity type PUBLISHER has an attribute Books published, then it can be calculated by counting the book entities associated with that publisher. Thus, the attribute is known as derived attribute. Similarly, consider an entity type PERSON and its attributes Date_of_birth and Age.

82%	MATCHING BLOCK 45/319	W
The value of the attribute Age can be determined from the current date and the value of		

Date_of_birth. Thus, the attribute Age is a derived attribute and the attribute Date_of_birth is known as stored attribute. ? Single-valued and multivalued attributes: The attributes that can have only one value for a given entity are called the single-valued attributes. For example, the attribute Book_title is a single-valued attribute as one book can have only one title. However, in some cases, an attribute can have multiple values for a given entity. Such attributes are called multivalued attributes. For example, the attributes Email_ID and Phone of the entity type PUBLISHER are multi-valued attributes, as a publisher can have zero, one or more e-mail IDs and phone numbers. The multivalued attributes can also have lower and upper bounds to restrict the number of values for each entity. For example, the lower bound of the attribute Email_ID can be specified to zero and upper bound to three. This implies that each publisher can have none or at most three e-mail IDs. ? Complex attributes: The attributes that are formed by arbitrarily nesting the composite and multivalued attributes are called complex attributes. The arbitrary nesting of attributes is represented by grouping components of a composite attribute between parentheses '()' and by displaying multivalued attributes between braces '{}'. These attributes are separated by commas. For example, a publisher can have offices at different places each with different address and each office can have multiple phones. Thus, an attribute Address_phone for a publisher can be specified as given in Figure 2.4. Here, Phone is a multivalued attribute and Address is a composite attribute. Fig. 2.4 Complex attribute — Address_phone Null Values for the Attributes Sometimes, there may be a situation where a particular entity may not have an appropriate value for an attribute. In such situations, the attribute takes a special value called null value. A null value of an attribute may indicate 'not applicable'. For example, consider an entity type PERSON with three attributes First_name, Middle_name and Last_name. Since, not every person has a middle name, the person without any middle name will have a null value for the attribute Middle_name. It implies that the attribute Middle_name is not applicable to that person. A null value can also be used in situations where the value of a particular attribute is unknown. An unknown category can further be classified as missing (the value that exists, but is not available) and not known (it is not known that the value exists or not). For example, if the attribute Book_title has the value null, it indicates that the title of the book is missing, as every book must have a title. However, if the attribute Phone has the null value, it indicates that it is not known whether the value for Phone exists or not. Key Attribute 28
The attribute (or combination of attributes) whose values are distinct for each individual instance of an entity type is known as a key attribute. It is used to uniquely identify each instance of an entity type. That is, no two entities can have the same value for the key attribute. For example, the attribute ISBN of the entity type BOOK is the key attribute, as two different books cannot have the same ISBN. An entity type may have more than one key attributes. For example, the attributes P_ID and Name of the entity type PUBLISHER are the key attributes, as no two publishers can have the same publisher ID and publisher name. Superkey, Candidate Key, and Primary Key A set of one or more attributes, when taken together, helps in uniquely identifying each entity is called a superkey. For example, the attribute P_ID of the entity type PUBLISHER can be used to uniquely identify each entity instance. Thus, P_ID is a superkey. Similarly, the attribute Name of the entity type PUBLISHER is also a superkey. The combination of the attributes P_ID and Name is also a superkey. One can also use the set of all attributes (P_ID, Name, Address, Phone, Email_ID) to uniquely identify each entity instance of type PUBLISHER. Thus, the combination of all attributes is also a superkey. However, P_ID itself can uniquely identify each entity instance, thus, other attributes are not required. Such a minimal superkey that does not contain any superfluous (extra) attributes in it is called a candidate key. A candidate key contains a minimized set of attributes that can be used to uniquely identify a single entity instance. For example, the attributes P_ID and Name are the candidate keys. However, the combination of P_ID and Name is not a candidate key. The candidate key, which is chosen by the database designer to uniquely identify entities, is known as the primary key. For example, the attribute P_ID can be chosen as the primary key. If a primary key is formed by the combination of two or more attributes, it is known as composite key. Note that a composite key must be minimal, that is, all attributes that form the composite key must be included in the key attribute to uniquely identify the entities. An entity type can have only one primary key; however, it can have multiple super keys and candidate keys. The attributes whose values are never or rarely changed are chosen as primary key. For example, the attribute Address cannot be chosen as a part of primary key as it is likely to be changed. Moreover, the attributes, which can have null values, cannot be taken as primary key. For example, the attributes Phone and Email_ID can have null values, as it is not necessary for every publisher to have an email ID or phone number. Weak and Strong

87%	MATCHING BLOCK 47/319	W
Entity Types An entity type that does not have any key attribute		

of its own is called a weak entity type. On the other hand, an entity type that has a key attribute is called a strong entity type. The weak entity is also called a dependent entity as it depends on another entity for its identification. The strong entity is called an independent entity, as it does not rely on another entity for its identification. The weak entity has no meaning and relevance in an E-R diagram if shown without the entity on which it depends. For example, consider an entity type EDITION with attributes edition no and Type (Indian or International). It depends on another entity type BOOK for its existence, as an edition cannot exist without a book. Thus, EDITION is a weak entity type and BOOK is a strong entity type. A weak entity type does not have its own identifying attribute that can uniquely identify each entity instance. It has a partial identifying attribute, which is known as partial key (or discriminator). A

partial key is a set of attributes that can uniquely identify

the instances of a weak entity type that are related to the same instance of a strong entity type. For example, the attribute Edition no of the entity type EDITION can be taken as a partial key.

The primary key of a weak entity type is formed by the

combination of its discriminator and the primary key of the strong entity type on which it depends for its existence. Thus, the primary key of the entity type EDITION is (ISBN, Edition_no). Since, the entities belonging to a weak entity type are identified by the attributes of a strong entity type in combination of one of their attribute, the strong entity types are known as identifying or parent entity type. The weak entity type, on the other hand, is known as child or subordinate entity type. 2.1.2 Relationships An association between two or more entities is known as a relationship. A relationship describes how two or more entities are related to each other. For example, the relationship PUBLISHED_BY (shown in Figure 2.5) associates a book with its publisher. Like entity types and entity sets, relationship types and a relationship sets also exist. Consider n possibly distinct 29
entity types E 1 , E 2 ,..., E n (where, n &lt; = 2). A relationship type R among these n entity types defines a set of associations (known as relationship set) among instances from these entity types. Like entity type and entity set, the relationship type and relationship set are also referred by the same name, say R. A relationship instance represents an association between individual entity instances. For example, an association between the entity types BOOK and PUBLISHER represents a relationship type; however, an association between the instances C++ and P001 represents a relationship instance. Each relationship instance includes only one instance of each entity type that is participating in the relationship. Mathematically, a relationship type R can be defined as a subset of the Cartesian product E 1 × E 2 × ... × E n . Similarly, the relationship set R can be defined as a set of relationship instances r i , where each r j associates n individual entity instances e 1 , e 2 , ..., e n of each entity type E 1 , E 2 , ..., E n , respectively. Each of the individual entity types E 1 , E 2 , ..., E n is said to participate in the relationship type R and each of the individual entity instances e 1 , e 2 , ..., e n is said to participate in the relationship instance r i . Figure 2.5 shows a relationship type PUBLISHED_BY between two entity types BOOK and PUBLISHER, which associates each book with its publisher. Here, r 1 , r 2 , r 3 , ..., r n denote relationship instances. Fig. 2.5 Relationship PUBLISHED_BY between two entity types BOOK and PUBLISHER A relationship type may also have descriptive attributes. Consider two entity types AUTHOR and BOOK, which are associated with the relationship type REVIEWS. Each author reviews some books and gives rating to that book. An attribute Rating could be associated with the relationship to specify the rating of the book given by the author (see Figure 2.6). Fig. 2.6 Relationship type REVIEWS with attribute Rating Identifying Relationship The relationship between a weak entity type and its identifying entity type is known as identifying relationship of the weak entity type. For example, consider the entity types EDITION (weak entity type) and BOOK (strong entity type) and a relationship type HAS that exists between them. It specifies that each book has an edition. Since, an edition cannot exist 30

without a book, the relationship type HAS is an identifying relationship as it associates a weak entity type with its identifying entity type. Relationship Degree The number of entity types participating in a relationship type determines the degree of the relationship type. The relationship that involves just one entity type is called unary relationship. In other words, the relationship that exists between the instances of the same entity type is a unary relationship. The relationship between two entity types is known as binary relationship and the relationship between three entity types is known as ternary relationship. In general, the relationship between n entity types is known as n-ary relationship. Thus, a unary relationship has a degree 1, a binary relationship has a degree 2, a ternary relationship has a degree 3, and an n-ary relationship has a degree n. The binary relationships are the most common relationships. For example, the relationship type PUBLISHED_BY between the entity types BOOK and PUBLISHER is a binary relationship. Role Names and Recursive Relationships Each entity type that participates in a relationship type plays a specific function or a role in the relationship. The role of each participating entity type is represented by the role name. If the participating entity types are distinct, their role names are implicit and are not usually specified. For example, in PUBLISHED_BY relationship, the entity type BOOK plays the role of a book and the entity type PUBLISHER plays the role of a publisher. Thus, the role names in this relationship are implicit. Now, consider another entity AUTHOR which represents the authors who can write books as well as review the books written by other authors and give feedbacks to them. Thus, an author can be a writer as well as a reviewer. Thus, in a relationship type FEEDBACKS (given in Figure 2.7), the entity type AUTHOR plays two different roles (author and a reviewer). Such a relationship is called a recursive relationship. In such relationships, it is necessary to explicitly specify the role names to distinguish the meaning of each participation. Figure 2.7 shows a recursive relationship FEEDBACKS, where each relationship instance r i has two lines each marked by 'r' for the role of a reviewer and 'w' for the role of an author, respectively. Here, the entity instance a 1 plays the role of a reviewer for the instances a 2 , a 3 , and a 4 and the instance a 2 plays the role of a reviewer for the instance a 5 . Fig. 2.7 Recursive relationship A recursive relationship is always a unary relationship as it represents relationship between the instances of the same entity type. Constraints on Relationship Type Each relationship type has certain constraints that restrict the possible combination of instances participating in the relationship set. The constraints on the relationship type are of two types, namely, mapping cardinalities and participation constraint. These two constraints are collectively known as structural constraints of a relationship type. 31

Mapping Cardinalities The maximum number of instances of one entity type to which an instance of another entity type can relate to is expressed by the mapping cardinalities or cardinality ratio. The mapping cardinalities can be used to describe the relationships between two or more entities. However, they are most commonly used to describe the binary relationships. For a binary relationship between two entity types E 1 and E 2 , the mapping cardinalities can be of four types (see Figure 2.8). Learn More Like a binary relationship, a ternary relationship also has four types of mapping cardinalities, namely, one-to-one-to-one (1 : 1 : 1),

60%	MATCHING BLOCK 49/319	SA	DBMS.docx (D68067493)

one-to-one-to-many (1 : 1 : M) or many-to-one-to-one (M : 1 : 1), one-to-many-to-many (1 : M : N)

or many-to- many-to-one (M : N : 1), and many-to-many-to-many (M : N : P). ? One-to-one: In one-to-one mapping, each instance of entity type E 1 is associated with at most one instance of entity type E 2 and vice-versa. It is represented by 1 : 1 relationship. For example, consider two entities PERSON and DRIVING_LICENSE. Each person can only have one driving license, and one driving license with a particular number can only be issued to a single person. That is, no two persons can have the same driving license. Thus, these two entities have 1 : 1 relationship [see Figure 2.8(a)]. A one-to-one relationship is generally implemented by combining the attributes of both the entity types into one entity type. In this example, the DRIVING_LICENCE being a weak entity type can be specified as one of the attributes of the entity type PERSON. ? One-to-many: In one-to-many mapping, each instance of entity type E 1 can be associated with zero or more (any number of) instances of type E 2 . However, an instance of type E 2 can only be associated to at most one instance of E 1 . It is represented by 1 : M relationship. For example, one publisher can publish many books; however, one book (with a particular ISBN) can only be published by one publisher. Thus, the entities PUBLISHER and BOOK have 1 : M relationship [see Figure 2.8(b)]. ? Many-to-one: In many-to-one mapping, each instance of entity type E 1 can be associated with at most one instance of type E 2 . However, an instance of type E 2 can be associated with zero or more (any number of) instances of type E 1 . It is represented by M : 1 relationship. For example, different books can be published by one publisher; however, one book can only be published by one publisher. Thus, the entities BOOK and PUBLISHER have M : 1 relationship [see Figure 2.8(c)]. NOTE A many-to-one relationship is same as one-to-many; however, from a different point of view. ? Many-to-many: In many-to-many mapping, each instance of entity type E 1 can be associated with zero or more (any number of) instances of type E 2 and an instance of type E 2 can also be associated with zero or more (any number of) instances of type E 1 . It is represented by M : N relationship. For example, one author can write many books and one book can be written by more than one authors. Thus, the entities AUTHOR and BOOK have M : N relationship [see Figure 2.8(d)]. Participation Constraint The participation constraint specifies whether an entity instance can exist without participating in a relationship with another entity. In some notations, this constraint is also known as minimum cardinality constraint. The participation constraints can be of two types, namely, total and partial. If every instance of entity type E participates in at least one relationship instance of type R, the participation is said to be total. On the other hand, if only some of the instances of entity type E participate in the relationship, the participation is said to be partial. 32 Fig. 2.8 Mapping cardinalities The total participation constraint is also known as mandatory participation, which specifies that an entity instance cannot exist without participating in the relationship. It implies that the participation of each entity instance in a relationship type is mandatory. For example, the participation of each instance of the entity type BOOK in the relationship type PUBLISHED_BY is total, as each book must be published by a publisher. On the other hand, the partial participation constraint, also known as optional participation, specifies that an entity instance can exist without participating in the relationship. It implies that the participation of entity instance in a relationship type is optional. For example, the participation of the entity PUBLISHER in the relationship type PUBLISHED_BY is partial, as not all publishers publish books —some may publish only journals, magazines, and newspapers. A weak entity type always has a total participation constraint, as each instance of weak entity type must be related to at least one of the instances of its parent entity type. 2.2 ENTITY-RELATIONSHIP DIAGRAM Once the entity types, relationships types, and their corresponding attributes have been identified, the next step is to graphically represent these components using entity-relationship (E-R) diagram. An E-R diagram is a specialized graphical tool that demonstrates the interrelationships among various entities of a database. It is used to represent the overall logical structure of the database. While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changed rarely; however, the instances in the entity and relationship sets change frequently. Thus, E-R diagrams are more useful in designing the database. An E-R diagram serves several purposes listed as follows: Learn More Entity-relationship models can be classified in two types, namely, Binary Entity Relationship Model (BERM) and General Entity Relationship Model (GERM). In a BERM, only binary relationships are allowed; however, in GERM, relationships between three or more entities ('n-ary' relationships) are also allowed. ? It is used to communicate the logical structure of the database to the end users. ? It helps the database designer in understanding the information to be contained in the database. ? It serves as a documentation tool. 2.2.1 E-R Diagram Notations There is no standard for representing the

76%	MATCHING BLOCK 50/319	SA	47F417BA15858476660.pdf (D123781188)

data objects in E-R diagrams. Each modeling methodology uses its own notation.

The original notation used by Chen is the most commonly used notation. It consists of a set of symbols that are used to 33

represent different types of information. The symbols include rectangles, ellipses, diamonds, lines, double lines, double ellipses, dashed ellipses, and double rectangles. All these symbols are described here and shown in Figure 2.9. ? The entity types such as BOOK, PUBLISHER, etc. are shown in rectangular boxes labelled with the name of the entity types. Weak entity types such as EDITION are shown in double rectangles. ? The relationship types such as PUBLISHED_BY, REVIEWS, FEEDBACKS, etc., are shown in diamond-shaped boxes labelled with the name of the relationship types. The identifying relationships such as HAS are shown in double diamonds. ? The attributes of the entity types such as Book_title, Price, Year, etc., are shown in ellipses. The key attributes are also shown in ellipses; however, with their names underlined. Multivalued attributes such as Email_ID and Phone are shown in double ellipses and derived attributes are shown in dashed ellipses. ? Straight lines are used to link attributes to entity types and entity types to relationship types. Double lines indicate the total participation of an entity type in a relationship type. ? The cardinality ratios of each binary relationship type are represented by specifying a 1, M or N on each participating edge. 2.2.2 E-R Diagram Naming Conventions Before initiating the designing of an E-R model, it is necessary to decide the format of the names to be given to the entity types, relationship types, attributes and roles, which will be used in the model. Some points must always be kept in mind while choosing the proper naming conventions. ? The names should be chosen in such a way that the meaning of the context is conveyed as much as possible. ? The usage of names should be consistent throughout the E-R diagram. ? The usage of hyphens, spaces, digits, and special characters should be avoided. ? The use of unpopular abbreviations and acronyms should be avoided. ? Singular names should be used for entity types. ? The binary relationship names should be chosen in such a way that makes the E-R diagram readable from left to right and from top to bottom. For example, consider the relationship type PUBLISHED_BY in Figure 2.12(a). This relationship type is read from left to right as 'the entity type BOOK is published by the entity type PUBLISHER'. However, if the position of the entities in the E-R diagram is changed, the relationship type PUBLISHED_BY becomes PUBLISHES and then it will be read as 'the entity type PUBLISHER publishes the entity type BOOK'. ? If a primary key of one entity type is used as a foreign key in another entity type, it should be used unchanged or with some extensions. Learn More There are many tools to construct E-R diagrams. Some of them are Avolution, ConceptDraw, ER/Studio, PowerDesigner, Rational Rose, SmartDraw, Microsoft Visio, and Visual Paradigm. 34
Fig. 2.9 E-R Diagram notations 2.2.3 E-R Diagram for Online Book Database As discussed in Chapter 01, the Online Book database maintains the details such as ISBN, title, price, year of publishing, number of pages, author and publisher of various textbooks, language books and novels. The user can also view the details of various authors and publishers. The author details include the name, address, URL, and phone number. The publisher details include the publisher name, address, e-mail ID, and phone number. The database also maintains the details of the ratings and feedbacks of other authors about a particular book. The ratings of a particular book are displayed along with the other details of the book. However, the detailed feedback about the book can be viewed on the URL of the authors who have reviewed the book. Based on this specification, the E-R diagram for Online Book database can be designed. This section discusses the representation of all the entity types, relationship types, attributes, role names and recursive relationships, weak entity types, and identifying relationship, and cardinality ratios and participation constraints of Online Book database in an E-R diagram. Representing Entity Types and Attributes The Online Book database consists of three entity types, namely, BOOK, PUBLISHER, and AUTHOR. The entity type BOOK consists of the attributes Book_title, Price, ISBN, Year, Page_count, and Category. The entity type PUBLISHER has the attributes P_ID, Name, Address, Phone, and Email_ID. The entity type AUTHOR has the attributes A_ID, Name, Address, URL, and Phone. The attributes ISBN, P_ID, and A_ID are the key attributes. All these entities with their corresponding attributes are shown in Figure 2.10 35
Fig. 2.10 Entity types of Online Book database Representing Multivalued, Composite, and Derived Attributes The attributes Phone and Email_ID are the multivalued attributes and Address is a composite attribute comprising House_number, Street, City, State and Zip_code as its component attributes. The Name of the author can also be taken as composite attribute with First_name, Middle_name, and Last_name as its component attributes. The composite attributes are shown by attaching ellipses containing the component attributes. Figure 2.11 shows the multivalued and composite attributes. It also shows some attributes in dashed ellipses. These attributes are the derived attributes. For example, the value of the attribute Books_published (for the PUBLISHER) can be calculated by counting the book entities associated with that publisher. Similarly, the attribute Books_written (for the AUTHOR) can be calculated by counting the book entities associated with that author. Fig. 2.11 Multivalued, composite, and derived attributes Representing Relationship Types and Their Attributes The entity types BOOK and PUBLISHER are associated with a relationship type PUBLISHED_BY by M : 1 relation. That is, many books can be published by one publisher; however, one book of a particular ISBN can be published by one publisher only. The entity types AUTHOR and BOOK are associated by the relation type WRITES by M : N relation. That is, one book can be written by many authors and one author can write many books. The entity types AUTHOR and BOOK are also associated with the relationship type REVIEWS by M : N relation, which specifies that one author can review many books and one book can be reviewed by many authors. 36

The participation (total or partial) of each entity in the relationship type can also be shown in an E-R diagram. For example, the participation of entity type BOOK in the relationship type PUBLISHED_BY is total, as every book must be published by a publisher. Similarly, the participation of entity types BOOK and AUTHOR in the relationship type WRITES is also total, as every book must be written by at least one author and each author must write at least one book. However, the participation of the entity types BOOK and AUTHOR in the relationship type REVIEWS is partial, as some books may not be reviewed by any author and some authors may not review any book. The E-R diagrams showing the relationship type PUBLISHED_BY, WRITES and REVIEWS, the participation of each entity in the relationship types, and their descriptive attributes are shown in Figure 2.12. If a relationship type has some descriptive attributes, then these attributes can also be shown in an E-R diagram. Figure 2.12 shows the relationship types PUBLISHED_BY and REVIEWS with the descriptive attributes Copyright_date and Rating, respectively. The attribute Copyright_date specifies the date on which the publisher gets the right of publishing the book for the first time. The attribute Rating contains a rated value (between 1 and 10) given to the book by other authors after reviewing the book. Fig. 2.12 E-R diagrams showing relationship between BOOK, AUTHOR, and PUBLISHER Representing Role Names and Recursive Relationships An E-R diagram can also be used to represent the role names and recursive relationships. For example, the author of one book acts as a reviewer for other books. Thus, a recursive relationship exists between the instances of the same entity type AUTHOR. This recursive relationship with the role names is shown in Figure 2.13. The role names (reviewer and author) are indicated by labelling the lines connecting the relationship types to entity types. Since, the relationship type FEEDBACKS involves only one entity type, it is a unary relationship. 37
Fig. 2.13 E-R diagram showing role names and recursive relationship Representing Weak Entity Types and Identifying Relationships In the Online Book database, the entity type EDITION is a weak entity type with Edition_no as its partial key. It depends on the strong entity type BOOK. It has a total participation in the identifying relationship type HAS, which specifies that an edition cannot exist without a book (see Figure 2.14). The complete E-R diagram for the Online Book database is shown in Figure 2.15. Fig. 2.14 E-R diagram showing weak entity, identifying relationship, and partial key Fig. 2.15 E-R diagram of Online Book database 2.2.4 E-R Diagram Representing Ternary Relationships The E-R diagram shown in Figure 2.15 includes only unary and binary relationships. An E-R diagram can also be used to represent higher degree relationships (relationships of degree greater than two). The most common higher degree relationship is ternary relationship, which describes the relationship between three entity types. For example, consider the entity types AUTHOR, BOOK, and SUBJECT. The entity type SUBJECT has the attributes S_ID and Subject_name. The subject name could be Database Systems, C++, C, Computer Architecture, etc. 38
Each author writes a book on a particular subject. This relationship is ternary as it associates three entity types. Figure 2.16 shows the relationship type WRITES among three entity types AUTHOR, BOOK, and SUBJECT. The mapping cardinality of the relationship type WRITES is M : 1 : 1, which implies that a group of particular authors (many authors) can write at most one book on a particular subject. Fig. 2.16 E-R Diagram showing a ternary relationship NOTE Relationships beyond ternary relationships are not very common; however, can sometimes exists. The discussion of the relationships greater than the degree three is beyond the scope of this book. 2.2.5 E-R Diagram—Design Choices While designing an E-R diagram, a number of design choices are to be considered. In other words, it is possible to represent a set of entities and their relationships in several ways. This section discusses each of these choices. Entity Types versus Attributes When the attributes of a particular entity type are identified, it becomes difficult to determine whether a property of the entity should be represented as an entity type or as an attribute. For example, consider the entity type AUTHOR with attributes A_ID, Name, Address, Phone, and URL. Treating phone as an attribute implies that each author can have at the most one telephone number. In case multiple phone numbers have to be stored for each author, the attribute Phone can be taken as a multivalued attribute. However, treating telephone number as a separate entity type better models a situation where the user may want to store some extra information such as location (home, office or mobile) of the telephone. Thus, Phone can be taken a separate entity type with attributes Phone_number and Location, which implies that an author can have multiple phone numbers at different locations. After making Phone as an entity type, the E-R diagram for the entity type AUTHOR can be redesigned as shown in Figure 2.17. In this figure, the entity types AUTHOR and PHONE are related with the relationship type HAS_PHONE with the cardinality ratio 1 : M. It implies that one author can have many phone numbers; however, one phone number can only be associated with one author. Fig. 2.17 E-R Diagram showing PHONE as an entity type Binary versus Ternary Relationship Generally, a database contains binary relationships, as it is easy to represent and understand the binary relationships. However, in some cases, ternary relationships need to be created to represent certain situations. For example, consider the ternary relationship WRITES in Figure 2.16. Its equivalent E-R diagram comprising three distinct binary relationships, 39

WRITES_ON, WRITES and OF with cardinality ratios M : N, M : N, and M : 1, respectively, is shown in Figure 2.18. By comparing Figures 2.16 and 2.18, it appears that the relationships between three entity types have changed from a single M : 1 : 1 to three binary relationships—two of them with M : N mapping cardinality and one with M : 1 mapping cardinality. Fig. 2.18 E-R Diagram showing three binary relationships The E-R diagram shown in Figure 2.16 conveys the meaning that a group of particular authors can write at most one book on a particular subject. That is, same group of authors cannot write many books on the same subject. On the other hand, the E-R diagram shown in Figure 2.18 conveys three different relationships. The relationship WRITES shows that one author can write many books and one book can be written by many authors. The relationship WRITES_ON shows that one author can write books on many subjects and books on one subject can be written by many authors. Finally, the relationship OF shows that many books can be written on a particular subject; however, one book can belong to only one subject. The information conveyed by these three binary relationships is not same as that of the ternary relationship shown in Figure 2.16. Thus, it is the responsibility of the database designer to make an appropriate decision depending on the situation being represented. 2.3 ENHANCED E-R MODEL

100%	MATCHING BLOCK 52/319	SA	BOOK SIZE.docx (D50092775)

The E-R diagrams discussed so far represents the basic concepts of a database schema. However, some aspects of a database such as inheritance among various entity types cannot be expressed using the basic E-R model. These aspects can be expressed by enhancing the E-R model. The resulting diagrams are known as enhanced E-R or EER diagrams and the model is called EER model. The basic E-R model can represent the traditional database applications such as typical data processing application of an organization effectively. On the other hand, the EER model is used to represent the new and complex database applications such as telecommunications, Geographical Information Systems (GIS), etc. This section discusses the extended E-R features including specialization, generalization, and aggregation and their representation using EER diagrams. 2.3.1

Specialization and Generalization In some situations, an entity type may include sub-groupings of its entities in such a way that entities of one subgroup are distinct in some way from the entities of other subgroups. For example, the entity type BOOK can be classified further into three types, namely, TEXTBOOK, LANGUAGE_BOOK, and NOVEL. These entity

89%	MATCHING BLOCK 51/319	W

types are described by a set of attributes that includes all the attributes of

the entity type BOOK and some additional set of attributes that differentiate them from each other. These additional attributes are also known as local or specific attributes. For example, the entity type TEXTBOOK may have the additional attribute Subject (for example, Computer, Maths, Science, etc.), LANGUAGE_BOOK may have the attribute Language (for example, French, German, Japanese, etc.), and the entity type NOVEL may have the attribute Type (Fiction, Mystery, Fantasy, etc.). This process of defining the subgroups of a given entity type is called specialization. The entity type containing the common attributes is known as the superclass and the entity type, which is a subset of the superclass, is known as its subclass. For example, the entity type BOOK is a superclass and the entity types TEXTBOOK, 40

LANGUAGE_BOOK, and NOVEL are its subclasses. This process of refining the higher-level entity types (superclass) into lower-level entity types (subclass) by adding some additional features to each of them is a top-down design approach. The design process may also follow a bottom-up approach in which multiple lower-level entity types are combined on the basis of common features to form higher-level entity types. For example, the database designer may first identify the entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL and then combine the common attributes of these entity types to form a higher-level entity type BOOK. This process is known as generalization. In simple terms, generalization is the reverse of specialization. The two approaches are different in terms of their starting and ending point. Specialization starts with a single higher-level entity type and ends with a set of lower-level entity types having some additional attributes that distinguish them from each other. Generalization, on the other hand, starts with the identification of a number of lower-level entity types and ends with the grouping of the common attributes to form a single higher-level entity type. Generalization represents the similarities among lower-level entity types; however, suppresses their differences. Specialization and generalization can be represented graphically with the help of an EER diagram in which the superclass is connected with a line to a circle, which in turn is connected by a line to each subclass that has been defined (see Figure 2.19). The '∪' shaped symbol on each line connecting a subclass to the circle indicates that the subclass is a subset '⊂' of the superclass. The circle can be ' '(empty) or it can contain a symbol d (for disjointness) or o (for overlapping), which is explained later in this section. Fig. 2.19 Specialization and generalization Attribute Inheritance As discussed earlier, the higher-level and lower-level entity types are created on the basis of their attributes. The higher-level entity type (or superclass) has the attributes that are common to all of its lower-level entity types (or subclasses). These common attributes of the superclass are inherited by all its subclasses. This property is known as attribute inheritance. For example, the attributes ISBN, Book_title, Category, Price, Page_count, Year, and P_ID of the entity type BOOK are inherited by the entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL. This inheritance of entity types represents a hierarchy of classes. Note that all the lower-level entity types of a particular higher-level entity types also participate in all the relationship types in which the higher-level entity type participates. For example, the entity type BOOK participates in the relationship type PUBLISHED_BY, its subclasses TEXTBOOK, LANGUAGE_BOOK, and NOVEL also participate in this relationship type. That is, all textbooks, language books, and novels are published by a publisher. All the attributes and relationship types of a higher-level entity type are also applied to its lower-level entity types but not vice-versa. 41
If a subclass is a subset of only one superclass, that is, it has only one parent, it is known as single inheritance and the resulting structure is known as a specialization (or generalization) hierarchy. For example, Figure 2.19 shows a single inheritance since the three subclasses are derived from only one superclass. On the other hand, if a subclass is derived from more than one superclass, it is known as multiple inheritance and the resulting structure is known as a specialization (or generalization) lattice. In this case, the subclass is known as a shared subclass. For example, consider two entity types EMPLOYEE and STUDENT. A student who is working as a part-time employee in an organization as well as pursuing a course from a university can be derived from these two entity types. Figure 2.20 shows a specialization lattice with a shared subclass PARTTIME_EMPLOYEE. Fig. 2.20 Specialization lattice with shared subclass PARTTIME_EMPLOYEE Constraints on Specialization and Generalization To design an accurate and effective model for a particular organization, some constraints can be placed on specialization or generalization. All the constraints can be applied to both generalization and specialization. However, in this section the constraints are discussed in terms of specialization only. Constraints on Subclass Membership In general, a single higher-level entity type (superclass) may have several lower-level entity types (subclasses). Thus, it is necessary to determine which entities of the superclass can be the members of a given subclass. The membership of the higher-level entities in the lower-level entity types can be either condition-defined or user-defined. ? Condition-defined: In condition-defined, the membership of entities is determined by placing a condition on the value of some attribute of the superclass. The subclasses formed on the basis of the specified condition are called condition-defined (or predicate-defined) subclasses. For example, all the instances of type BOOK can be evaluated based on the value of the attribute Category. The entities that satisfy the condition Category = "Textbook" are allowed to belong to the subclass TEXTBOOK only. Similarly, the entities that satisfy the condition Category = "Language Book" are allowed to belong to the subclass LANGUAGE_BOOK only. Lastly, the entities that satisfy the condition Category = "Novel" are allowed to belong to the NOVEL subclass only. Thus, these conditions are the constraints that determine which entities of the superclass can belong to a given subclass. If the membership condition for all subclasses in specialization is defined on the same attribute (in this case, Category), it is known as

100%	MATCHING BLOCK 53/319	SA	BOOK SIZE.docx (D50092775)

an attribute-defined specialization, and the attribute is called the defining attribute

of the specialization. The condition-defined subclass is shown

83%	MATCHING BLOCK 55/319	SA	BOOK SIZE.docx (D50092775)

by writing the condition next to the line connecting the subclass to the specialization circle (

see Figure 2.21). The symbol d denotes the disjointness constraint. ? User-defined: If the membership of the superclass entities in a given subclass is determined by the database users and not by any membership condition, it is called user-defined specialization, and the subclasses that are formed are known as user-defined subclasses. For example, consider an entity CELEBRITY. A celebrity could be a film star, a politician, a newsreader or a player. Thus, the assignment of the entities of type CELEBRITY to a given subclass is specified explicitly by the database users when they apply the operation to add an entity to the subclass (see Figure 2.22). The symbol o denotes the overlapping constraint. 42
Fig. 2.21 EER diagram for an attribute-defined specialization with disjoint constraint Fig. 2.22 EER diagram for a user-defined specialization with overlapping constraint Disjoint and Overlapping Constraints In addition to subclass membership constraints, two other types of constraints, namely, disjointness and overlapping, can also be applied to a specialization. These constraints determine whether a higher-level entity instance can

87%	MATCHING BLOCK 54/319	W

belong to more than one lower-level entity type within a single

specialization or not. ? Disjointness constraint: This constraint specifies that the same higher-level entity instance cannot belong to more than one lower-level entity types. That is, the subclasses of any superclass must be disjoint. For example, an entity of type BOOK can belong to either TEXTBOOK or NOVEL but not both. An attribute-defined specialization in which the defining attribute is single-valued implies a disjointness constraint. The disjoint constraint is represented by a symbol d written in a circle in an EER diagram as shown in Figure 2.21. ? Overlapping constraint: This constraint specifies that the same higher-level entity instance can belong to more than one lower-level entity types. That is, the subclasses of any superclass need not to be disjointed and entities may overlap. In terms of EER diagram, the overlapping constraint is represented by a symbol o written in a circle joining the superclass with its subclasses. For example, the entity types PLAYER and POLITICIAN show an overlapping constraint, as the celebrity can be a player as well as a politician (see Figure 2.22). Similarly, an entity of type BOOK can belong to both TEXTBOOK and LANGUAGE_BOOK since a language book can also be a prescribed textbook in a university [see Figure 2.23(b)]. NOTE The lower-level entity types are by default overlapped; however, the disjointness constraint must be explicitly placed on generalization and specialization. Completeness Constraint The last constraint that can be applied to generalization or specialization is the

81%	MATCHING BLOCK 59/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

completeness constraint. It determines whether an entity in higher-level entity set must belong to at least one of the lower-level entity sets

or not. The completeness constraint can be total or partial. 43
? Total specialization: It specifies that each

77%	MATCHING BLOCK 56/319	W

higher-level entity must belong to at least one of the lower-level entity types in the specialization.

Figure 2.23(a) shows the total specialization of the entity type BOOK. Here, each book entity must belong to either textbook or language book or novel category. The total specialization is represented by double lines connecting the superclass with the circle. ? Partial Specialization: It allows some of the instances of higher-level entity type not to belong to any of the lower- level entity types. Figure 2.23(b) shows the partial specialization of the entity type BOOK, as all books do not necessarily belong to the category textbooks or language books, some may belong to the category novels also. 2.3.2 Aggregation The E-R diagrams discussed so far represent the relationships between two or more entities. An E-R diagram cannot represent the relationships among relationships. However, in some situation, it is necessary to represent a relationship among relationships. The best way to represent these types of situations is aggregation. The process through which one can treat the relationships as higher-level entities is known as aggregation. For example, in the Online Book database, the relationship WRITES between the entity types AUTHOR and BOOK can be treated as a single higher-level entity called WRITES. The relationship WRITES and the entity types AUTHOR and BOOK are aggregated into a single entity type to show the fact that once the author has written a book then only it gets published. The relationship type PUBLISHED_BY can be shown between the entity types PUBLISHER and WRITES as shown in Figure 2.24. The relationship type PUBLISHED_BY is a many-to-one relationship. It implies that a book written by a group of authors can be published by only one publisher; however, one publisher can print many books written by different authors. Fig. 2.23 EER diagrams showing completeness constraints 44
Fig. 2.24 EER diagram for Online Book database showing aggregation Note that the EER diagram shown in Figure 2.24 cannot be expressed as ternary relationship between three entity types AUTHOR, BOOK, and PUBLISHER, as WRITES and PUBLISHED_BY are two different relationship types and hence, cannot be combined into one. 2.4 ALTERNATIVE NOTATIONS FOR E-R DIAGRAMS An E-R diagram can also be represented using some other diagrammatic notations. One alternative E-R notation allows specifying the cardinality of the relationship, that is, the number of entities that can participate in a relationship. In this notation, a pair of integers (min, max) is associated with

each participation of an entity type E in a relationship type R,

where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$. This implies that each entity instance e of type

E must participate in at least min and at most max relationship instances in R.

If min is specified as zero, it means partial participation and if min &lt; 0, it means total participation. Figure 2.25 shows the modified E-R diagram of Online Book database with (min, max) notation. For example, the cardinality (1, N) of the AUTHOR implies that an author must write at least one book; however, an author can write any number of books. Similarly, the cardinality (1, 4) for the BOOK implies that a book must be written by at least one author and maximum by four authors. Fig. 2.25 E-R diagram for Online Book database with (min, max) notation 45

Another notation is crow's feet notation in which the symbol used to represent the many side of the relationship resembles the forward digits of a bird's claw. Figure 2.26 shows the modified E-R diagram of Online Book database with crow's feet notation. It actually consists of three symbols, namely, empty circle 'O', vertical bar '|' and crow's feet '&gt;'. These symbols can be used in four combinations listed as follows: ? &gt; + O = optional many (either zero, one or more) ? | + &gt; = mandatory many (at least one or many) Fig. 2.26 E-R diagram Online Book database with crow's feet notation ? | + | = mandatory one (one and only one) ? | + O = optional one (either zero or one but not many) 2.5 UNIFIED MODELING LANGUAGE E-R diagrams only represent various entities of an organization and relationships among them. The entities and their relationships form only one part of the overall software design. A software design includes some other components also such as functional modules of the system and interactions among them, user interactions with the system, etc. These components can be specified using a standard called the Unified Modeling Language (UML), which was officially defined by the Object Management Group (OMG). Learn More Rational Rose is currently a popular tool used to draw UML diagrams. It helps the software developers in designing clear and easy-to-understand UML models. Unified Modeling Language (UML) is used as one of the techniques to implement object data modeling. The main advantage of object data modeling is that it models the elements of the system as real-world objects and hence, represents the real world very closely. Initially, it was designed for modeling object-oriented software; however, nowadays it is also used for business process modeling, systems engineering modeling, and representing organizational structures. 2.5.1 UML Diagrams UML includes a number of diagrams that are used to create an abstract model of a system. This abstract model is usually known as a UML model. UML includes ten standard diagrams, which are divided into two main categories, namely, structural diagrams and behavioural diagrams. Figure 2.27 shows the hierarchy of these UML diagrams. 46
Fig. 2.27 Hierarchy of UML diagrams Structural Diagrams The structural diagrams describe the static or structural relationships among various components of the system. The structural diagrams are of five types. ? Class diagram: A class diagram describes the static structure of the entire system by showing the classes of the system, their attributes, and the relationships between them. Like E-R diagrams, a UML class diagram is also used to represent the conceptual database schema. The entity types in an E-R diagram become the classes in the UML class diagram. ? Package diagram: A package diagram can be treated as a subset of a class diagram. It organizes the elements of a system into related groups such that the dependencies between the various packages is minimized. ? Object diagram: An object diagram describes a set of objects (instances of a class) and their relationships. It is used to represent the static view of a system at a particular point of time and is typically used to test the class diagrams for accuracy. ? Component diagram: A component diagram displays the dependencies among various components of the system. The components include source code, run-time (binary) code, and executable code. ? Deployment diagram: A deployment diagram displays the physical resources in the system such as nodes, components, and connections. The nodes are the run-time processing elements. Behavioural Diagrams The behavioural diagrams describe the dynamic or behavioural relationships among various components of the system. The behavioural diagrams are of four types. ? Use case diagram: A use case diagram displays the relationship among actors and use cases. An actor represents a user or another system that interacts with the system and a use case is an external view of the system that represents some action. ? Statechart diagram: A statechart diagram describes the dynamic behaviour of the system by displaying the sequences of states that an object goes through during its life in response to some external events. It shows the start/initial state of an object before the occurrence of an external event and its stop/final state in response to that external event. ? Activity diagram: An activity diagram describes the dynamic behaviour of the system by modeling the flow of control from one activity to another. An activity is an operation performed on any class in the system that results in a change in the system state. ? Interaction diagrams: An interaction diagram models the dynamic characteristics of a system by representing the set of messages exchanged among a set of objects. They are further divided into two categories. 47

o Sequence diagram: It describes the interactions among classes in terms of an exchange of messages over time. It consists of the vertical and horizontal dimensions, where vertical dimension represents the time and horizontal dimension represents different objects participating in the interactions. o Collaboration diagram: It represents the interactions among various objects in terms of a series of sequenced messages. In a collaboration diagrams, the objects are shown as icons and the messages are numbered to show a particular sequence of messages. In sequence diagrams the emphasis is on the time-ordering of messages; however, in collaboration diagrams the emphasis is on structural organization of objects that interacts with each other by sending messages. 2.5.2 Representing Conceptual Schema Using Class Diagrams In a class diagram each entity type of an E-R diagram becomes the class and is displayed as a box containing three sections. The top section contains the class name, the middle section contains the attributes of the class and the last section contains the operations that can be performed on the objects of the class. A binary relationship type is represented by just drawing a line connecting the participating classes. The name of the relationship type is written adjacent to the line. The role names can also be specified by writing the role name on the line. If the relationship type has descriptive attributes, they are shown in a box containing the name of the relationship type. This box is connected to the relationship line by a dashed line. The (min, max) notation is represented as min .. max notation. For example, the cardinality (1, 1) is shown as 1 .. 1, (1, N) is shown as 1 .. *, and (0, N) is shown as * or 0..*. Here, the symbol * indicates no maximum limit on participation. Figure 2.28 shows the UML class diagram for the Online Book database. Fig. 2.28 UML class diagram for the Online Book database Representing Specialization and Generalization In UML class diagrams, specialization or generalization is represented by connecting the subclasses with the superclass by a blank or filled triangle. A blank triangle indicates specialization with disjoint constraint and a filled triangle indicates specialization with overlapping constraint. Figure 2.29 shows UML class diagrams representing specialization and generalization with disjoint and overlapping constraint. Note that both single and multiple inheirtance can be shown using UML class diagram notation. The topmost superclass is known as base class and the bottommost subclasses are known as leaf classes. 48
Fig. 2.29 Specialization and generalization in UML class diagram SUMMARY 1. The conceptual modeling (or semantic modeling) is an important phase in designing a successful database. It represents various pieces of data and their relationships at a very high-level of abstraction. It mainly

100%	MATCHING BLOCK 60/319	SA	47F417BA15858476660.pdf (D123781188)
focuses on what data is required and how it should be organized rather than what operations			

are to be performed on the data. 2. The conceptual model can be represented using two major approaches, namely, entity-relationship (E-R) modeling and the object modeling. 49
3. The E-R model

100%	MATCHING BLOCK 61/319	SA	Database System.pdf (D165747767)
views the real world as a set of basic objects (			

known as entities), and their characteristics (known as attributes) and associations among these objects (known as relationships). 4. An entity is a distinguishable object

68%	MATCHING BLOCK 63/319	SA	Database System.pdf (D165747767)
that has an independent existence in the real world. If an entity has a physical existence,			

it is termed as tangible or concrete entity and if it has a conceptual existence, it is termed as non- tangible or abstract entity. 5. Each entity is represented by a set of attributes. The attributes are the properties of an entity that characterize and describe it. 6. Each attribute can accept a value from a set of permitted values, which is called the domain or the value set of the attribute. 7. A set or a collection of entities that share the same attributes but different values is known as an entity type. 8. A specific occurrence of an entity type is called its instance. The collection of all instances of a particular entity type in the database at any point of time is known as an entity set. 9. The attributes of an entity are classified into five categories, namely, identifying and descriptive attributes, simple and composite attributes, stored, and derived attributes, single and multivalued attributes, and complex attributes. 10. Sometimes, there may be a situation where a particular entity may not have an appropriate value for an attribute. In such situations, the attribute takes a special value called null value. 11. The attribute (or combination of attributes), whose values are distinct for each individual instance of an entity type is known as a key attribute. 12. A set of one or more attributes, taken together, that helps in uniquely identifying each entity is called a superkey. 13. A minimal superkey that does not contain any superfluous (extra) attribute in it is called a candidate key. 14. The candidate key, which is chosen by the database designer to uniquely identify entities, is known as the primary key. 15. If a primary key is formed by the combination of two or more attributes it is known as a composite key. 16. An entity type that does not have any key attributes of its own is called a weak entity type. On the other hand,

an entity type that has a key attribute is called a strong entity type. 17. A weak entity

type has a partial identifying attribute known as partial key (or discriminator), which is a set of attributes that can uniquely identify the instances of a weak entity type related to the same instance of a strong entity type. 18. An association between two or more entities is known as a relationship. The relationship between a weak entity type and its identifying entity type is known as an identifying relationship of the weak entity type. 19. The number of entity types participating in a relationship type determines the degree of the relationship type. 20. Each entity type that participates in a relationship type plays a specific function or a role in the relationship. The role of each participating entity type is represented by the role name. 21. Each relationship type has certain constraints that restrict the possible combination of instances participating in the relationship set. The constraints on the relationship type are of two types, namely, mapping cardinalities and participation constraint. 22. An E-R diagram is a specialized graphical tool that is used to represent the overall logical structure of the database. 23. An E-R diagram consists of a set of symbols that are used to represent different types of information. The symbols include rectangles, ellipses, diamonds, lines, double lines, double ellipses, dashed ellipses, and double rectangles. 24. An enhanced E-R diagram is used to represent

some other aspects of a database such as inheritance among various entity types. 25.

The process of defining the subgroups of a given entity type is called specialization. The entity type containing the common attributes is known as the superclass and the entity type, which is a subset of the superclass, is known as its subclass. 26. The design process in which multiple lower-level entity types are combined on the basis of common features to form higher-level entity types is known as generalization. 27. If a subclass is a subset of only one superclass, that is, it has only one parent, it is known as single inheritance and the resulting structure is known as a specialization (or generalization) hierarchy. On the other hand, if a subclass is 50
derived from more than one superclass, it is known as multiple inheritance and the resulting structure is known as a specialization (or generalization) lattice. 28. The process through which one can treat the relationships as higher-level entities is known as aggregation. 29. Unified modeling language (UML) is used to specify some other components of a software design process including functional modules of the system and interactions among them, user interactions with the system, etc. 30. UML includes ten standard diagrams that are divided into two main categories, namely, structural diagrams and behavioural diagrams. 51

CHAPTER 3 THE RELATIONAL MODEL After reading this chapter, the reader will understand: ? The relational model and its three components, namely, structural component, manipulative component and integrity constraints. ? The relation schema and relation instance ? Attribute and its domain ? Unstructured and structured domain ? Degree and cardinality of a relation ? Characteristics of relations ? The relational database schema and relational database instance ? Keys including superkey, candidate key, primary key, alternate key and foreign key ? Various types of data integrity including domain integrity, entity integrity, referential integrity and semantic integrity ? Different types of integrity constraints ? Legal and illegal relation ? Constraint violation while insert, delete, and update operation ? Mapping of E-R model to relational model ? Representation of entity types to relation ? Representation of weak entity types, relationship types, composite and multivalued attributes, and Extended E-R features A database is a collection of interrelated data and the way data is related to each other depends upon the model being used. Generally databases are based on one of the three models: the hierarchical model, the network model or the relational model. A database based on the network or hierarchical model is a non-relational database and a database based on the relational model is a relational database. A relational database is a collection of relations (or two- dimensional tables) having distinct names. It is a persistent storage mechanism that conforms to the relational model. At the time when the relational model was proposed, hierarchical and network model were used to structure the data. Both these models are complex since they use pointers to relate the records. However, the relational model is simple as it uses the values of the records instead of pointers to relate the records. For example, in Online Book database, the common value from the publisher and book records such as publisher id is used to relate the publisher record to the book record, instead of using an internal pointer. In addition, it gives the freedom to change the way of storing and accessing the data without corresponding change in the way the user perceives the data. The relational model is based on the branches of mathematics, namely, set theory and predicate logic. The key assumption is that the data is represented in the form of mathematical n-ary relations, which is a subset of the Cartesian product of n sets. Reasoning about such a data is done in two-valued predicate logic in mathematical model. In two-valued predicate logic, two possible evaluations for each proposition are possible, that is, either true or false. In addition to true or false, third value such as null is included in three-value predicate. Some consider only two-valued logic as crucial part of the relational model, whereas others include three-valued logic as crucial part of the relational model. The relational model includes three components, namely, structural component (or data structures), manipulative component (or data manipulation), and integrity constraints. Structural component consists of two components, namely, entity types and their relationships, which act as a framework of the given situation. Manipulative component consists of a set of operations that are performed on a table or group of tables. Integrity constraints are a set of rules that governs the entity types and their relationships, and thereby, ensures the integrity of database. 3.1 RELATIONAL MODEL CONCEPTS The relational model represents both data and the relationships among those data using relation. A relation is used to represent information about any entity (such as book, publisher, author, etc.) and its relationship with other entities in the 52
form of attributes (or columns) and tuples (or rows). A relation comprises of a relation schema and a relation instance. A relation schema (also termed as relation intension) depicts the attributes of the table, and a relation instance (also termed as relation extension) is a two-dimensional table with a time-varying set of tuples. Relation Schema A relation schema consists of a relation name R and a set of attributes (or fields) A 1 , A 2 , ..., A n . It is represented by R(A 1 , A 2 , ..., A n) and is used to describe a relation R. To understand the concept of relation schema, consider the book, publisher, and author information stored in an Online Book database with the schema of the relations given as
BOOK(ISBN, Book_title, Category, Price, Copyright_date, Year, Page_count, P_ID)
AUTHOR(A_ID, Aname, City, State, Zip, Phone, URL) PUBLISHER(P_ID, Pname, Address, State, Phone, Email_id) Learn More Codd's model emerged as the dominant model that provides the basis for effectively using both the relational database software, such as Oracle, MS SQL Server, etc., and the personal database systems such as Access, FoxPro, etc. The main basis in developing relational model is that a database is a collection of unordered tables that can be modified through various operations that return a table or group of tables as their result. In these statements,
ISBN, Book_title, Category, Price, Copyright_date, Year, Page_count,
and P_ID are the attributes of relation BOOK. A_ID, Aname, City, State, Zip, Phone, and URL are the attributes of relation AUTHOR. P_ID, Pname, Address, State, Phone, and Email_id are the attributes of relation PUBLISHER. An attribute can be single-valued as well as multivalued attribute. Single-valued attributes represent single value of data, whereas multivalued attributes represent more than one value of data. Examples of single-valued attributes of relation PUBLISHER are P_ID, Pname, Address, and State, whereas Phone and Email_id are multivalued attributes of relation PUBLISHER. An attribute consists of an attribute name and a value of one or more bytes from the set of permissible values of same data type. This set of permissible value of same data type for an attribute is termed as the domain D of that attribute. The relation schema of book information can be written using the data type of each attribute as BOOK(
ISBN: string, Book_title: string, Category: string, Price: integer, Copyright_date: integer, Year: integer, Page_count: integer, P_ID: string) In this statement, the

permissible value for the attribute ISBN consists of a set of character values. Thus, the domain for the attribute ISBN is string. Note that the data type or format for each domain is the physical definition of domains and the value of the domain is the logical definition of domains. Here, the data type string is the physical definition of domain and a set of character values is the logical definition of domain. Several attributes may have same domain at the physical level but distinct domains at the logical level. For example, both the attributes Aname and Pname of relations AUTHOR and PUBLISHER, respectively, have same data type, that is, string. Hence, both the attributes have same domain at the physical level. However, both the attributes Aname and Pname have different values, say, Thomas William and Hills Publications, respectively. Hence, both the attributes have distinct domains at the logical level. Several attributes may have either same or distinct domains at both physical and logical level. For example, the attributes State of relations AUTHOR and PUBLISHER have same domain both at the physical level, that is string, and at the logical level, say, New York. On the other hand, the attributes URL and Phone of relations AUTHOR and PUBLISHER, respectively, ought to have distinct domains both at the physical level that is, string for URL and integer for Phone, and also at the logical level. Note that domain can be unstructured or structured. Unstructured (or atomic) domain consists of the non-decomposable or indivisible values that cannot be further divided into sub-values. Atomic domains are also known as application-independent domains since it consists of the general sets that are independent of a particular application. Sets of strings, integers, real numbers, etc. are some of the examples of unstructured domain. On the other hand, structured (or composite) domain consists of non-atomic values that are decomposable or divisible values. The domain for the attribute 53
Address of relation PUBLISHER consisting of house number, street name, and city is an example of structured domain. Structured domains are also known as application-dependent domains since these are the subset of standard data types and can be defined by specifying the constraints on the values that can be inserted into the application. Relation Instance A relation instance (

68%	MATCHING BLOCK 65/319	SA	BOOK SIZE.docx (D50092775)

or relation) r of the relation schema R(A 1 , A 2 , ..., A n) is a set of n-tuples t.

It is also denoted by r(R). A relation instance is an ordered set of attributes values v that belongs to the domain D and it can be denoted as r = {t 1 , t 2 , ..., t m } where, t = {v 1 , ..., v n }. In other words, it is a set of tuples (or records) having the same number of attributes as the relation schema. It can be specified as t r, which denotes that tuple t is in relation r. A relation instance can be considered as a table, which is a collection of tuples having the same number of attributes. Note that a relation with n attributes is a subset of a Cartesian product of n domains. In other words, a relation of n attributes is a subset of D 1 × D 2 × D 3 × ... × D n . Consider the BOOK relation in which domains D 1 , D 2 , D 3 , ..., D 8 denote the set of all ISBN, Book_title, Price,...,P_ID, respectively. Now any tuple of BOOK relation must comprise of eight attributes (A 1 , A 2 , ..., A 8), where the values of A 1 (ISBN) belong to domain D 1 and similarly, the values of remaining attributes A 2 , A 3 , ..., A 8 belong to domains D 2 , D 3 , ..., D 8 , respectively. Since a BOOK relation is a subset of all possible attributes values in a list of domains, it can be written as a subset of D 1 × D 2 × D 3 × ... × D 8 . Consider the BOOK relation of Figure 3.1, the instance B1 contains nine tuples and eight attributes. If the tuple t denotes the first tuple of the relation, the notation t [ISBN] refers to the value of t on the ISBN attribute. Hence, t [ISBN] = "001- 354-921-1", t [Book_title] = "Ransack" and so on. Mathematically, it can also be written as t [1] = "001-354-921-1", t [2] = "Ransack" and so on. Here, 1 refers to the first attribute of a BOOK relation, 2 refers to the second attribute, and so on. Fig. 3.1 Instance B1 of the BOOK relation The number of attributes in a relation is known as degree or arity, and

100%	MATCHING BLOCK 66/319	SA	BOOK SIZE.docx (D50092775)

the number of tuples in a relation is known as cardinality.

A relation of degree one is known as unary relation, of degree two is known as binary relation, of degree three is known as ternary relation, and of degree n is known as n-ary relation. For example, the degree of the relation in Figure 3.1 is eight and its cardinality is nine. 3.1.1 Characteristics of Relations 54
A relation has certain characteristics, which are given here. Learn More A relation of zero degree (or with no attributes) is known as nullary relation and it is of the type RELATION{}. There are two relations of zero degree, namely, TABLE_DEE (or DEE) and TABLE_DUM (or DUM). DEE contains only one tuple, namely, 0- tuple and is of the type RELATION{TUPLE {}}. On the other hand, DUM is empty and contains no tuples. DEE and DUM are mainly used in the relational algebra and they represent TRUE (or YES) and FALSE (or NO), respectively. ?

56%	MATCHING BLOCK 67/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

Ordering of Tuples in a Relation: Since a relation is a set of tuples and a set has no particular order among its elements, tuples in a relation do not have any specified order. However, tuples in a relation

can be logically ordered by the values of various attributes. In that case, information in a relation remains the same, only the order of tuple varies. Hence, tuple ordering in a relation is irrelevant. For example, if the BOOK relation in Figure 3.1 is logically ordered by the values of attribute Category, the first tuple of BOOK relation in Figure 3.1 becomes the third tuple of BOOK relation in Figure 3.2. Similarly, second tuple becomes the fifth tuple in Figure 3.2. Thus, every tuple of BOOK relation in Figure 3.1 is the tuple of BOOK relation in Figure 3.2 and vice versa. Since both the relations contain the same set of tuples, they are the same. Fig. 3.2 The BOOK relation (sort by Category) ? Ordering of values within a Tuple: As discussed earlier, n-tuple is an ordered set of attributes values that belongs to the domain D, so, the order in which the values appear in the tuples is significant. However, if a tuple is defined as a set of (&gt;attribute&lt;: &gt;value&lt;) pairs, the order in which attributes appear in a tuple is irrelevant. This is due to the reason that there is no preference for one attribute value over another. For example, consider these statements. t =&gt;
ISBN: 001-987-760-9, Book_title: C++, Category: Textbook, Price: 40, Copyright_date: 2003, Year: 2005, Page_count: 800, P_ID:
P001 &lt; 55
t =&gt;Category: Textbook, ISBN: 001-987-760-9, Price: 40, P_ID: P001, Book_title: C++, Page_count: 800, Copyright_date: 2003, Year: 2005 &lt; In these statements, since the attributes and values are represented as pairs, these two tuples are the same. ? Values and Nulls in the Tuples: Relational model is based on the assumption that the tuples in a relation contain only atomic values. In other words, each tuple in a relation contains a single value for each of its attribute. Hence, a relation does not allow composite and multivalued attributes. Moreover, it allows denoting the value of the attribute as null, if the value does not exist for that attribute or the value is unknown. For example, if a publisher does not have an email address, the null value can be specified, which signifies that the value does not exist (see Figure 3.3). Fig. 3.3 The PUBLISHER relation NOTE A relation that assigns a single value to each attribute is said to be in first normal form, which will be discussed in Chapter 06. ? No two tuples are identical in a relation: Since a relation is a set of tuples and a set does not have identical elements. Therefore, each tuple in a relation must be uniquely identified by its contents. In other words, two tuples with the same value for all the attributes (that is, duplicate tuples) cannot exist in a relation. ? Interpretation of a Relation: A relation can be used to interpret facts about entities as a type of assertion. For example, the PUBLISHER relation in Figure 3.3 asserts that an entity PUBLISHER has P_ID, Pname, Address, State, Phone, and Email_id. Here, the first tuple asserts the fact that there is a publisher whose publisher id is P001, name is Hills Publications, and so on. A relation can also be used to interpret facts about relationships. For example, consider the BOOK relation in Figure 3.1, the attribute P_ID relates the BOOK relation with the PUBLISHER relation. Learn More E.F. Codd designed a set of 13 rules (numbered 0 to 12), known as Codd's 12 rules, to define relational database system. These rules are so strict that almost all the popular RDBMSs fail to meet the criteria. 3.2 RELATIONAL DATABASE SCHEMA A set of relation schemas {R 1 , R 2 , ..., R m } together with a set of integrity constraints in the database constitutes relational database schema. A relational database schema, say S, is represented as S ={R 1 , R 2 , ..., R m). For example, the relational database schema Online Book is a set of relation schemas, namely, BOOK, PUBLISHER, AUTHOR, AUTHOR_BOOK, and REVIEW, which is shown in Figure 3.4. 56

Fig. 3.4 Relational database schema Hence, the relational database schema Online Book can be represented as Online Book ={BOOK, PUBLISHER, AUTHOR, AUTHOR_BOOK, REVIEW} NOTE To define a relational database schema, a non-procedural language, SQL, is required to communicate with relational database. SQL will be discussed in Chapter 05. 3.3 RELATIONAL DATABASE INSTANCE A relational database instance s of relational database schema S ={R 1 , R 2 , ..., R m } is a set of relation instances {r 1 , r 2 , ..., r m }such that each relation instance r i is a state of corresponding relation schema R i . In addition, each relation instance must satisfy the integrity constraints specified in the relational database schema. For example, a relational database instance corresponding to the Online Book schema is shown in Figure 3.5. Note that when relational model was developed, some of its versions were developed on the assumption that those attributes which represent same real-world concept would be given same attribute names in all relations. This assumption had a limitation that it would not be possible to represent attributes having same real-world concept with different meanings in the same relation. Nowadays, different names can be given to the attributes representing the same real-world concept with different meanings in the same relation. To elaborate this concept, consider the modified form of Online Book database in which the relations, namely, AUTHOR_BOOK and REVIEW are merged together in a single relation say, AUTHOR_REVIEWER. The resultant relation is shown in Figure 3.6. In the AUTHOR_REVIEWER relation, the concept of author id appears twice, once as an author and another as a reviewer who is also an author. The different attribute names, that is, A_ID and R_ID are given to them to represent different meaning in the same relation. Attributes representing the same real-world concept may have same or different names in different relations. For example, the attribute P_ID in both PUBLISHER and BOOK relations represents the same concept with same attribute name in different relations. However, the attribute A_ID in the AUTHOR relation, and the attribute R_ID in the REVIEW relation represent the same concept, but with different attribute name in different relations. On the other hand, attributes representing different real-world concept may have same name in different relations. For example, the attribute Pname in the PUBLISHER relation and the attribute Aname in the AUTHOR relation represents different real-world concept, that is, publisher name and author name. However, they can be given a same attribute name, say, Name in different relations, that is, PUBLISHER and AUTHOR relations. 57
58
Fig. 3.5 Relational database instance 59
Fig. 3.6 The AUTHOR_REVIEWER relation 3.4 KEYS Key is one of the important concepts of relational database. It can be superkey, candidate key, primary key, and foreign key. Superkey A superkey (SK) is a subset of attributes of a relation schema R, such that for any two distinct tuples t 1 and t 2 in the relation state r of R, we have t 1 [SK] ≠ t 2 [SK]. It means that the combination of attributes in SK uniquely identify each tuple in r. In every relation schema R, we have at least one default superkey, that is, the set of all the attributes, since no two tuples can be identical in all of its attributes. For example, consider a relation schema BOOK with three attributes ISBN, Book_title, and Category. As we know, the value of ISBN is unique for each tuple; hence, {ISBN} is a superkey. In addition, the combination of all the attributes, that is, {ISBN, Book_title, Category} is a default superkey for this relation schema. Candidate Key Generally, all the attributes of a superkey are not required to identify each tuple uniquely in a relation. Instead, only a subset of attributes of the superkey is sufficient to uniquely identify each tuple. Further, if any attribute is removed from this subset, the remaining set of attributes can no longer serve as a superkey. Such a minimal set of attributes, say K, is a candidate key (also known as irreducible superkey). For example, the superkey {ISBN, Book_title, Category} is not a candidate key, since its subset {ISBN} is a minimal set of attributes that uniquely identify each tuple of BOOK relation. So, ISBN is a candidate key. Note that each relation can have more than one candidate key, and all candidate keys are super-keys whereas all superkeys are not candidate keys. Further, if a superkey includes a single attribute, it is a candidate key. However, if a candidate key has multiple attributes, all these attributes must be needed to uniquely identify each tuple. 60

Primary Key Since a relation can have more than one candidate key, one of these keys has to be chosen as the primary key to uniquely identify each tuple. A candidate key that is chosen to uniquely identify a tuple in a relation is known as primary key (PK). Other candidate keys that are not chosen as primary key are known as alternate keys. For example, in BOOK relation, there are two candidate keys, namely, ISBN and Book_title (considering no two books can have the same title). If the attribute ISBN is chosen as a primary key, the attribute Book_title becomes the alternate key. Generally, the primary key comprises one attribute in a relation; however, more than one attribute can form the primary key. In that case, primary key is known as composite key. The attribute whose value is never or rarely changed should be chosen as primary key. For example, in PUBLISHER relation, the attribute Address should not be chosen as a primary key since it can be changed often. However, the attribute P_ID can be chosen as a primary key since each publisher has a unique publisher id and generally, it does not change. Foreign Key Attribute of one relation can be accessed in another relation by enforcing a link between the attributes of two relations. This can be done by defining the attribute of one relation as the foreign key that refers to the primary key of another relation. For example, consider two relations, namely, PUBLISHER and BOOK. Here, all the books must be associated with a publisher that is already in the PUBLISHER relation. In this case, a foreign key will be defined on the BOOK relation, which will be related to the primary key of the PUBLISHER relation (see Figure 3.7). Thus, all books in the BOOK relation would be related to a publisher in the PUBLISHER relation. Things to Remember The candidate key with least number of attributes should be chosen as the primary key of the relation. A relation that references another relation is known as referencing relation, whereas a relation that is being referenced is known as referenced relation. For example, in Figure 3.7, the PUBLISHER relation is a referenced relation and the BOOK relation is a referencing relation. A foreign key may refer to its own relation. Such a foreign key is known as self-referencing or recursive foreign key. A relation can have zero or more foreign keys and each foreign key can refer to different referenced relations. For example, consider REVIEW relation that includes the details of ratings given by the reviewer to a particular book. Here, books and reviewers (or authors) must be associated with book and author that are already in the BOOK and AUTHOR relations, respectively. In this case, foreign keys will be defined on the REVIEW relation that will be related to the primary keys of the BOOK and AUTHOR relations (see Figure 3.8). 61 Fig. 3.7 Foreign key Fig. 3.8 More than one foreign key in one referencing relation A foreign key attribute in referencing relation may have a different name from that of the primary key attribute of referenced relation. However, the domain of both the attributes must be same. For example, the foreign key attribute R_ID of the REVIEW relation has a different name from the primary key attribute A_ID of the referenced relation AUTHOR. However, the data type of both the attributes is same, that is, string. A

80%	MATCHING BLOCK 70/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

foreign key in a referencing relation that matches a primary key in a referenced relation

represents many-to-one relationship between referencing and referenced relation. 3.5 DATA INTEGRITY The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity
ensures that the data in the database is consistent, accurate, correct, and valid.
It ascertains that the data adhere to the set of rules defined by the database administrator and hence, prevents the entry of the invalid information into database. Data integrity is of four types, namely, domain integrity, entity integrity, referential integrity, and semantic integrity. Generally, domain integrity is applied on the attribute; entity integrity is applied on the tuple; referential integrity is applied on the relation; and semantic integrity ensures logical data in the database. This section discusses the data integrity in detail. Domain Integrity Domain integrity can be specified for each attribute by defining its specific range or domain. It requires that the attribute value must fall under a particular range in order to be valid. It ensures correct values for an attribute by defining the restrictions on the data type, format or range of possible values. For example, the domain of the attribute Price of the BOOK relation for the Category: Textbook may be between $20 and $200. Similarly, the domain of the attribute Category may be Textbook, Novel or Language Book. More formally, domain integrity ensures A in D where, A denotes an attribute, which has the same set of values as that of the domain D. Note that a tuple with n attributes satisfying domain integrity can be written as {&gt; A 1 :V 1 ,A 2 :V 2 ,..., A n :V n &lt;|V 1 ∊D 1 ,V 2 ∊D 2 ,...,V n ∊D n } where, 62

A 1 , A 2 , ..., A n are attributes of a tuple V 1 , V 2 , ..., V n are the value set associated with the domain D 1 , D 2 , ..., D n , respectively, These statements imply that the attribute value must belong to its associated domain values. For example, the first tuple of PUBLISHER relation can be written as &gt;P_ID: P001, Pname: Hills Publications, Address: 12, Park Street, Atlanta, State: Georgia, Phone: 7134019, Email_id: h_pub@hills.com &lt; Note that the domain integrity tests the value entered by the user into the database by ensuring that the attribute value entered by the user is in its associated domain. The attribute values that do not violate any domain integrity rules are considered as valid values even if they are logically incorrect. For example, if the Price attribute of the BOOK relation is assigned a value 65 instead of 56, this value is considered as valid since it belongs to the set of values defined in the domain, that is, between $20 and $200. Instead of defining a set of range for the attribute, domain integrity can be defined conditionally on the attribute values that can be validated according to the respective rules applied on them. If the attribute values entered by the user violate the domain integrity rules, the entered values are rejected. For example, the restriction on the domain of the attribute Price of BOOK relation can be applied as If Category is Textbook, Price must be between $20 and $200 If Category is Language Book, Price must be between $20 and $150 If Category is Novel, Price must be between $20 and $100 Entity Integrity Entity integrity assigns restriction on the tuples of a relation and ascertains the accuracy and consistency of the database. It ensures that each tuple in a relation is uniquely identified in order to retrieve each tuple separately. The primary key of a relation is used to identify each tuple of a relation uniquely. Entity integrity is a rule, which ensures that the set of attribute(s) that participates in the primary key cannot have duplicate value or null value. This is because each tuple in a relation is uniquely identified by a primary key attribute and if the value of the primary key attribute is duplicate or null, it becomes difficult to identify such tuple uniquely. For example, if the attribute ISBN is declared as a primary key of BOOK relation, entity integrity ensures that any two tuples of BOOK relation must have unique value for the attribute ISBN and must not be null. NOTE If an attribute is declared as a primary key, it adheres to entity integrity rule. Entity integrity rule requires that various operations like insert, delete, and update on a relation maintains uniqueness and existence of a primary key. In other words, insert, delete, and update operations on a relation results in valid values, that is, unique and non-null values in primary key attribute. Any operation that provides either duplicate or null value in primary key attribute is rejected. Note that it is not mandatory for each relation to have a primary key but it is a good practice to create a primary key for each relation Referential Integrity Referential integrity condition ensures that the domain for a given set of attributes, say S1, in a relation r1 should be the values that appear in certain set of attributes, say S2, in another relation r2. Here, the set S1 is foreign key for the referencing relation r1 and S2 is primary key for referenced relation r2. In other words, the value of
foreign key in the referencing relation must exist in the primary key attribute of the referenced relation, or that the value of foreign key is null. Referential integrity also ensures that the data type of the foreign key in referencing relation must match the data type of the primary key of referenced relation. For example, the data type of both foreign key attribute P_ID in BOOK relation and primary key attribute P_ID in PUBLISHER relation is same, that is, string. 63 Note that there may be some tuples in the referenced relation that do not exist in the referencing relation. For example, in Figure 3.9, the tuple with the attribute value P005 in PUBLISHER relation does not exist in the BOOK relation. Fig. 3.9 Referential integrity Semantic Integrity To represent real world accurately and consistently, the business rules and logical rules (that are derived from our knowledge of the semantics of the real world) must be enforced on database. These rules are known as semantic integrity. Semantic integrity ensures that the data in the database is logically consistent and complete with respect to the real world. Examples of semantic integrity are: ? Number of pages of a book cannot be zero. ? One book can be published by one publisher only. ? Copyright date of a book cannot be later than its published date. ? An author cannot review his own book. 3.5.1 Enforcing Data Integrity In order to maintain the accuracy and consistency of data in a database, integrity of the data is enforced through integrity constraints. Integrity constraints are the rules defined on a relational database schema and satisfied by all the instances of database in order to model the real-world concepts correctly. They ensure the consistency of database while the modifications are made by the users. For example, the integrity constraint ensures that the value of ISBN in BOOK relation cannot be duplicated or null. A relation that conforms to all the integrity constraints of the schema is known as legal relation. On the other hand, a relation that does not conform to all the integrity constraints of the schema is known as illegal relation. Integrity constraints can be categorized into three parts, namely, table-related constraints, domain constraints, and assertion constraints. ? Table-related constraints are defined within a relation definition. This type of constraint can be specified either as a part of the attribute definition or as an element in the relation definition. When a constraint is specified as a part of the attribute definition, it is known as column-level constraint. Column-level constraints are checked when the constrained attribute is modified. On the other hand, when a constraint is specified as an element in the relation definition, it is known as table-level constraint. Table-level constraints are checked when any attribute of a relation is modified. Both column-level and table-level constraints hold different types of constraints (see Figure 3.10). 64

Fig. 3.10 Table-related constraints ? Domain constraints are specified within a domain definition. In other words, it is associated with any attribute that is defined within a particular domain. It supports not null and check constraints. ? Assertion constraints are specified within an assertion definition, which will be discussed in Chapter 05. NOTE The syntax for enforcing various constraints is given in Chapter 05. Primary Key Constraint The primary key constraint ensures that the attributes, which are declared as primary key must be unique and not null. Primary key constraint enforces entity integrity for the relation and constraints the relation in the following ways: ? The primary key attribute must have unique set of values for all the tuples in a relation. ? The primary key attribute cannot have null value. ? A relation can have only one
primary key attribute. Unique Constraint The unique constraint ensures that a particular set of attributes contains unique values
and hence, two tuples cannot have duplicate values in specified attributes. Like primary key, it also enforces entity integrity. For example, the unique constraint can be specified on Pname attribute of PUBLISHER relation to prevent tuples with duplicate publisher names. Unique constraint is preferable over primary key constraint to enforce uniqueness of a non-primary key attribute. This is because ? Multiple unique constraints can be applied on a relation, whereas only one primary key constraint can be defined on a relation. ? Unlike primary key constraint, unique constraint allows the attribute value to be null. Primary key and unique key constraints act as a parent key when applying a foreign key constraint. Parent key is the key that identifies the tuples of the relation uniquely. Check Constraint The check constraint can be used to restrict an attribute to have values in a given range, that is, for each tuple in a relation, the values in a certain attribute, must satisfy a given condition. One way to achieve this is by applying check constraint during the declaration of the relation. For example, if the check constraint (PRICE&lt;20) is applied on the attribute Price of the BOOK relation, it ensures that the value entered into the attribute Price must be greater than $20.
Check constraints can be applied on more than one attribute simultaneously. For example, in the attributes Copyright_date
and Published_date, constraints can be applied in such a way
that Copyright_date must be either less than or equal to the
Published_date. NOTE If the inserted value violates the check constraint, the insert or update operations will not be permitted. The check constraint can also be applied during domain declaration.
It allows specifying a condition that must be satisfied by any value assigned to an attribute whose type is the domain. For example, the check constraint can ensure that the 65
domain, say dom, allows values
between 20
and 200. If domain dom is assigned to the attribute Price of a BOOK relation, it ensures that the attribute Price must have values between 20 and 200. As a result, if a value that is not between 20 and 200 is inserted into the constrained attribute, an error occurs.
Not Null Constraint Every attribute has a nullability characteristic that shows whether a particular attribute accepts a null value or not. By default, every attribute accepts null values but this default nullability characteristic can be overridden by using the
not null constraint. The not null constraint ensures that the attribute must not accept null values. For example, consider a tuple in the BOOK relation where the attribute ISBN contains a null value. Such a tuple provides book information
for an unknown ISBN and hence, it does not provide appropriate information. In such case, null value must not be allowed and this can be done by constraining the attribute ISBN using not null constraint. Here, the not null constraint prevents the null value to be inserted into the attribute ISBN. Any attempt to change the attribute value of ISBN to null value results in an error. Note that the null values are not allowed in the primary key attribute of a relation. Since ISBN is a primary key attribute of BOOK relation, it cannot accept null values. Hence, it is not necessary to explicitly specify not null constraint for the attribute ISBN. The not null constraint enforces domain integrity by ensuring
that the attribute of a particular domain is not permitted to take null values. For example, a domain, say dom2, can be restricted to take non-null values.
If the domain dom2 is assigned to the attribute Category of a BOOK relation, it ensures that the attribute Category must have some values. As a result, if null value is inserted into the constrained attribute,

it will not be accepted as it violates the not null constraint. Foreign Key Constraint A foreign key constraint allows certain attributes in one relation to refer to attributes in another relation. The relation on which foreign key constraint is defined contains the partial information. Its detailed information can be searched from another relation with a matching entry. In this case, a foreign key constraint will not allow the deletion of a tuple from the relation with detailed information if there is a tuple in the relation with partial information exist. In other words, a foreign key constraint not only controls the data that can be stored in the referencing relation but it also controls the changes made to the referenced relation. For example, if the tuple for a publisher is deleted from the PUBLISHER relation, and the publisher's ID is used in the BOOK relation, the deleted publisher's ID becomes orphaned in the BOOK relation. This situation can be prevented by a foreign key constraint. This constraint ensures that changes made to the data in the PUBLISHER relation will not make the data in the BOOK relation orphan. Foreign key constraint enforces referential integrity by ensuring that an attribute in a relation S whose value in each tuple either matches with the values of the primary key in another relation R or is null. For example, the foreign key attribute R_ID in the REVIEW relation must have either the same values as that of the primary key A_ID of the AUTHOR relation or must be null. Learn More It is not necessary that the foreign key constraint can only be related with the primary key constraint in another relation. A foreign key constraint can also be defined to refer to the attributes of a unique constraint in another relation. 3.6 CONSTRAINT VIOLATION WHILE UPDATING DATA In addition to storing the data, various operations such as add, remove, and modify can also be performed on the relation. These operations must be performed in such a way that various integrity constraints should not be violated. The update operations on the relation are insert, delete, and update (or modify). 3.6.1 The Insert Operation The insert operation creates a new tuple or a set of tuples into a relation and provides attribute values to the inserted tuples. These inserted values must satisfy all the integrity constraints to maintain the consistency of the database. For 66
example, the statement Insert &gt;'003-456-654-3', 'Discrete Mathematics', 'Textbook', 60, 2007, 2007, 650, 'P002' &lt; satisfies all the integrity constraints and hence, these values can be inserted into the BOOK relation. However, if the attribute values violate one or more integrity constraints, the attribute values are rejected and the insert operation cannot be performed. Consider some cases given here. ? If the attribute value is inserted into a tuple of BOOK relation, it must belong to the associated domain of possible values otherwise the domain integrity will be violated. For example, in the statement, Insert &gt;'002-678-999-5', 'LINUX', 'Textbook', 300, 2006, 2007, 750, 'P005'&lt; into BOOK, the price of the book does not belong to the specified domain, that is, between $20 and $200. So, this statement violates the domain integrity and hence, the values are rejected. ? If the attribute value is inserted into a primary key attribute of BOOK relation, it must be unique and non-null otherwise the entity integrity will be violated. For example, in the statement, Insert &gt;'003-456-654-3', 'Software Engineering', 'Textbook', 75, 2005, 2006, 700, 'P003'&lt; into BOOK, the ISBN of the book has the same value as the ISBN of the existing tuple. So, this statement violates the primary key constraint and hence, the values are rejected. Similarly, consider another statement, Insert &gt; null, 'Software Engineering', 'Textbook', 75, 2005, 2006, 700, 'P004'&lt; into BOOK. In this statement, the ISBN of the book does not have any value, so it violates the entity integrity and hence, the values are rejected. ? If the attribute value is inserted into a foreign key attribute of BOOK relation, it must exist in the PUBLISHER relation otherwise the referential integrity will be violated. For example, in the statement, Insert &gt;'004-765-558- 5', 'Introduction to Japanese Language', 'Language Book', 45, 2007, 2007, 550, 'P007'&lt; into BOOK, the P_ID of the BOOK relation does not exist in the P_ID of the PUBLISHER relation. So, this statement violates the referential integrity and hence, the values are rejected. ? If the attribute value is inserted into a foreign key attribute of BOOK relation, it must be logically correct otherwise the semantic integrity will be violated. For example, in the statement, Insert &gt;'004-765-558-5', 'Introduction to French Language', 'Language Book', 45, 2007, 2007, 0, 'P003'&lt; into BOOK, the number of pages is zero, which is logically incorrect. So, this statement violates the semantic integrity and hence, the values are rejected. 3.6.2 The Delete Operation A tuple or a set of tuples can be deleted from a relation using the delete operations. Special attention must be given to the tuples that are being referenced by other tuples in the database, since it can violate the referential integrity. If deletion violates the referential integrity then it is either rejected or cascaded. For example, if an attempt is made to delete the tuple with P_ID = "P004" in PUBLISHER relation, the delete operation is rejected as it violates the referential integrity. On the other hand, if the tuple of PUBLISHER relation with P_ID = "P005" is deleted, the attribute values of this tuple will be deleted since this tuple is not referenced by any tuple of BOOK relation and does not violate the referential integrity. Deletion can be cascaded by deleting the tuples in the referencing relation that references the tuple to be deleted in the referenced relation. For example, in order to delete P_ID = "P004" from PUBLISHER relation, the tuples with P_ID = "P004" of BOOK relation must be first deleted. In addition to rejecting and cascading the deletion, the tuples that reference the tuple to be deleted can be modified either to null value or to another existing tuple. Note that assigning the null value to the tuple must not violate the entity integrity. If it violates then the null value should not be assigned to the tuple. Note that it is not possible to cascade the deletion in each case. To understand this concept, consider the modified form of Online Book database in which the relation BOOK is altered by adding an attribute A_ID. The resultant relation can be given as shown in Figure 3.11. 67

Fig. 3.11 The modified BOOK relation If an attribute value A001 has to be deleted from the AUTHOR relation, the corresponding tuple with the attribute value A001 must also be deleted from the modified BOOK relation. If this is done, in that case the corresponding book as well as publisher information of that particular author will also be deleted. So in this case, the attribute value of the author (whose tuple has to be deleted) in the modified BOOK relation is updated to null value instead of deleting that tuple. 3.6.3 The Update Operation The attribute values in a tuple or a set of tuples can be modified using the update operation. Update operation changes the existing value of the attribute in a tuple to the new and valid value. It ensures that the new value satisfies all the integrity constraints. For example, if the value of the attribute Price with ISBN= "001-987-760-9" is updated from $25 to $50, the changes are reflected in a relation since it satisfies all the integrity constraints. Things to Remember If a referencing attribute is a part of the primary key and violates the referential integrity, we cannot set it to null value, since doing this would result in the violation of entity integrity. Changing the value of attributes does not create any problem as long as the valid values are entered in a relation. However, changing the value of primary key attribute or foreign key attribute can be a critical issue. Since the primary key attribute identifies each tuple, changing this attribute value must satisfy all the integrity constraints. For example, if the value of the attribute P_ID in PUBLISHER relation is updated from P001 to P002, the changes are reflected in a relation only if it satisfies all the integrity constraints. Similarly, the foreign key attribute must be modified in such a manner, which ensures that the new value either refers to the existing value in the referenced relation or is null. For example, if the P_ID of the BOOK relation with ISBN= "001-987- 760-9" is updated to null or any other value that exists in the attribute P_ID of PUBLISHER relation, the value of the foreign key attribute is modified. However, if the new value of P_ID of the BOOK relation does not exist in the P_ID of the PUBLISHER relation, it violates the referential integrity and hence, the values are rejected. 3.7 MAPPING E-R MODEL TO RELATIONAL MODEL E-R model provides a conceptual model of the real-world concepts, which is represented in a database. To represent the E- R database schema to the relation schema, the relational model is used. In other words, E-R model is mapped to the relational model by representing

100%	MATCHING BLOCK 68/319	W
E-R database schema by a collection of relation schemas. 68		

The mapping from E-R model to relational model is possible due to the reason that both the models represent the real world logically and follow similar design principles. The Online Book database is used as an example to illustrate the mapping from E-R database schema to a collection of relation schema. Each entity type and relationship type in the Online Book database is represented as a unique relation schema and relation to which the name of the corresponding entity type or relationship type is assigned. The E-R schema of Online Book database is shown in Figure 3.12. In this section, the steps to translate an E-R database schema into a collection of relation schema and relations are shown. Fig. 3.12 E-R schema for the Online Book database 3.7.1 Representation of Entity Types to Relation An entity type is mapped to a relation by representing different attributes A 1 , A 2 , ..., A n of an entity type E as attributes A 1 , A 2 , ..., A n of a relation R. Each entity instance of the entity type E represents a tuple in the relation R, and the primary key of the entity type E becomes the primary key of the relation R. Consider the E-R diagram of the entity type BOOK in Figure 3.13. The entity type BOOK has six attributes, namely, ISBN, Book_title, Category, Price, Year, and Page_count. Fig. 3.13 The BOOK entity type The entity type BOOK can be represented by a relation schema BOOK with six attributes BOOK(ISBN, Title, Category, Price, Year, Page_count) 69
As discussed earlier, the mapping from E-R model to relational model results in a relation. The resultant relation is known as entity relation since each entity instance of the entity type represents a tuple in a relation. A relation with six attributes of the BOOK schema is shown in Figure 3.14. Fig. 3.14 The BOOK relation In this figure, the tuple &gt;'001-354-921-1', 'Ransack', 'Novel', 22, 2006, 200 &lt; represents that the book with ISBN '001-354- 921-1' has a book title 'Ransack' under the category 'Novel' with 200 pages, its price is $22, and it is published in the year 2006. NOTE The attributes P_ID and Copyright_date will be added into the BOOK relation in later steps. 3.7.2 Representation of Weak Entity Types Representation of weak entity type with attributes a 1 , a 2 , ..., a m depends on the strong entity type with the attributes A 1 , A 2 , ..., A n . Weak entity type must have total participation in the relationship type and must participate in one-to-many relationship with the strong entity type that defines it (that is, one strong entity type with many weak entity types). Weak entity type can be represented by the

60%	MATCHING BLOCK 69/319	W
relation schema R with one attribute for each member of the set as {a 1 , a 2 , ..., a m } ∪ {A 1 , A 2 , ..., A n } The primary key		

of the weak entity type consists of the primary key of the strong entity type as foreign key and its own partial key. In addition to the primary key, foreign key constraints are applied on the

54%	MATCHING BLOCK 74/319	SA	BOOK SIZE.docx (D50092775)

weak entity type as the primary key of the weak entity type refers to the primary key of the strong entity

type. Whenever a strong entity type is deleted, the related weak entity types must also be deleted.. For example, consider the E-R diagram of entity type BOOK having weak entity type EDITION shown in Figure 3.15. The weak entity type EDITION has two attributes, namely, Edition_no and Type. 70
Fig. 3.15 The entity type BOOK with weak entity type EDITION The primary key of the entity type BOOK, on which weak entity type EDITION depends, is ISBN. Thus, the relation schema of the weak entity type EDITION can be represented as EDITION= (ISBN, Edition_no, Type) In this statement, the primary key consists of the primary key of BOOK, along with the partial key of Edition, which is Edition_no. The foreign key constraint can be created on the Edition schema with the attribute ISBN referencing the primary key attribute of the BOOK relation and Edition_no is its own partial key. The corresponding EDITION relation is shown in Figure 3.16. 3.7.3 Representation of Relationship Types The relationship type R is mapped to a relation, which includes ? Primary key of each participating entity type as a foreign key ? The set of descriptive attributes (if any) of R Generally, primary key attributes of the participating entity types are chosen to identify a specific tuple in a binary relationship types. The primary key is chosen as shown in Table 3.1. Consider the relationship type REVIEWS in the E-R diagram of Figure 3.17. Each author can review one or more books and similarly, one book can be reviewed by one or more authors. For example, the relationship type REVIEWS has

47%	MATCHING BLOCK 71/319	W

two entity types, namely, AUTHOR with the primary key A_ID, and BOOK with the primary key ISBN. Since the relationship type has

one descriptive attribute, namely, Rating, the REVIEW schema of the relationship type REVIEWS has three attributes that can be shown as REVIEW(R_ID, ISBN, Rating) As stated earlier, the same concept of author id can be given a different name in different relation in order to distinguish the role of a reviewer from the author. Hence, the attribute A_ID of the AUTHOR relation is given a different name, that is, R_ID in the REVIEW schema. ISBN Edition_no Type 001-354-921-1 First International 001-987-650-5 Second Limited 001-987-760-9 Third Limited 002-678-880-2 First International 002-678-980-4 Fourth International 003-456-433-6 First Limited 71
ISBN Edition_no Type 003-456-533-8 Second Limited 004-765-359-3 Second International 004-765-409-5 Third International Fig. 3.16 The EDITION relation Since

50%	MATCHING BLOCK 76/319	SA	BOOK SIZE.docx (D50092775)

the relationship type is many-to-many, the primary key for the REVIEW relation is the union of the primary key of

entity type AUTHOR, that is, R_ID and primary key of entity type BOOK, that is, ISBN. In addition, R_ID and ISBN cannot be null since these keys are primary keys and hence, not null constraint and unique constraint are implicit for these fields. In addition to the primary key, foreign key constraints are also created on the REVIEW relation with attribute A_ID referencing the primary key of AUTHOR relation and attribute ISBN referencing the primary key of BOOK relation. Table 3.1 Identifying primary key for binary relationship types Relationship Type Primary Key for Relation one-to-one Primary key of either entity type participating in the relationship type

57%	MATCHING BLOCK 72/319	W

one-to-many or many-to-one Primary key of entity type on the 'many' side of the relationship

type many-to-many
Union of primary key of each participating entity type

n-ary without arrow on its edges Union of primary key attributes from the participating entity types n-ary with arrow on one of its edges Primary key of the entity type not on the 'arrow' side of relationship

type Fig. 3.17 A relationship type REVIEWS The corresponding relation REVIEW of the relation schema is shown in the Figure 3.18. Note that in the AUTHOR relation, no reference to the BOOK relation is given. Similarly, reference of AUTHOR relation is not given in the BOOK relation. This is because, in case of many-to-many relationship, this is logically implemented by including a separate relation. In this case, a REVIEW relation is included that relates AUTHOR and BOOK relations. The REVIEW relation links the primary key of AUTHOR relation with the primary key of the BOOK relation. NOTE A relationship type relating a weak entity type to the corresponding strong entity type is not required in a relational database since it creates redundancy. R_ID ISBN Rating A001 002-678-980-4 2 A002 001-987-760-9 6 72 R_ID ISBN Rating

A003 002-678-980-4 5 A003 004-765-409-5 4 A004 003-456-533-8 9 A005 002-678-980-4 7 A006 001-354-921-1 7 A006 002-678-880-2 4 A007 004-765-359-3 3 A008 001-987-760-9 7 A009 001-987-650-5 8 A010 003-456-433-6 5

Fig. 3.18 The REVIEW relation So far, we map each relationship type R to a different schema. Alternatively, the schema of R can be merged with the schema of one of the entity type that participates in R. Consider a many-to-one relationship type PUBLISHED_BY shown in Figure 3.19. The double line in the Figure 3.19 indicates that the participation of BOOK in the relationship type PUBLISHED_BY is total. Hence, the book cannot exist without being associated with a particular publisher. Thus, we can combine the schema for PUBLISHED_BY with the schema of BOOK, and we require only two schemas, which are given here.
BOOK(ISBN, Book_title, Category, Price, Copy-right_date, Year, Page_count, P_ID)
PUBLISHER(P_ID, Pname, Address, State, Phone, Email_id) The corresponding relation BOOK can be given as shown in Figure 3.20. The primary key of entity type, into whose schema the relationship type schema is combined, becomes the primary key of combined schema. So, the primary key of the combined schema is the primary key of BOOK, that is, ISBN. The schema representing the relationship type would have foreign key constraints that refer to each of the participating entity type in the relationship type. Here, the foreign key constraint of the entity type, into whose schema the relationship type schema is combined, is dropped and foreign key constraints of the remaining entity types are added to the combined schema. For example, the foreign key constraint that refers to the BOOK is dropped, whereas the foreign key constraint with P_ID that refers to PUBLISHER is retained as a constraint on the combined schema BOOK. Fig. 3.19 E-R diagram of relationship type PUBLISHED_BY 73
Fig. 3.20 The BOOK relation NOTE Even if the participation of the entity type in the relationship type is partial, the schemas can be combined by using null values. For one-to-one relationship type, the schema of the relationship type can be merged with the relation schema of either of the entity types. 3.7.4 Representation of Composite and Multivalued Attributes As discussed in Section 3.6.1, different attributes of an entity type become the attributes of a relation. If an entity type has composite attributes, no separate attribute is created for the composite attribute itself. Instead, separate attributes for each of its component attributes are created. For example, consider the E-R diagram of the entity type AUTHOR with the composite attribute Address shown in Figure 3.21. Fig. 3.21 E-R diagram of entity type AUTHOR The attribute Address of the entity type AUTHOR is a composite attribute with the component attributes, namely, City, State, and Zip. The separate attributes are created for these component attributes and the schema of the AUTHOR relation is given here. 74

AUTHOR(A_ID, Aname, State, City, Zip, Phone, URL) For the multivalued attribute, a separate relation is created with the primary key of entity type and with an attribute corresponding to the multivalued attribute of the entity type. For example, the attribute Phone of entity type AUTHOR is a multivalued attribute. Consider the E-R diagram of multivalued attribute Phone of entity type AUTHOR in Figure 3.22. Fig. 3.22 The entity type AUTHOR with multivalued attributes Multivalued attribute Phone of entity type AUTHOR can be represented by the relation schema as AUTHOR_PHONE(A_ID,Phone) And the corresponding relation, say AUTHOR_PHONE, can be given as shown in Figure 3.23. A_ID Phone A001 923673 A001 923743 A002 376045 A002 376123 A003 419456 A003 419562 A004 578123 A004 578932 Fig. 3.23 The AUTHOR_PHONE relation Note that each value of the multivalued attribute Phone is represented in a separate tuple of the AUTHOR_PHONE relation. For example, an author with A_ID as A001 and Phone as 923673 and 92374 is represented in two tuples, namely, {A001, 923673} and {A001, 923743} in the AUTHOR_PHONE relation. If the multivalued attribute is an alternate

key of the entity type, the primary key of new relation will be the combination of the primary key and the multivalued attribute of the entity

type. 3.7.5
Representation of Extended E-R Features So far, we have discussed the representation of basic features of E-R model. In this section we discuss the representation of extended E-R features, which include generalization/specialization and aggregation. The representation of 75
generalization involves two different approaches of designing the relation. The first approach creates separate relation for the higher-level entity type as well as for all the lower-level entity types. The relation for higher-level entity type includes its primary key attributes and all other attributes. Each relation for lower-level entity type includes the primary key of higher-level entity type and its own attributes. The primary key attribute of lower-level entity type refers to the primary key attribute of the higher-level entity type and hence, creates the foreign key constraint. For example, consider the EER diagram of entity type BOOK representing generalization in Figure 3.24. Fig. 3.24 EER diagram representing generalization In this figure, the higher-level entity type is BOOK and the lower-level entity types are TEXTBOOK, LANGUAGE_BOOK, and NOVEL. The relation schemas for the higher-level and lower-level entity types can be given as BOOK(ISBN, Book_title, Category, Price, Year, Page_count) TEXTBOOK(ISBN, Subject) LANGUAGE_BOOK(ISBN, Language) NOVEL(ISBN, Type) The corresponding relation BOOK is shown in Figure 3.14 and corresponding relations TEXTBOOK, LANGUAGE_BOOK, and NOVEL are shown in Figure 3.25. In this figure, the primary key attribute ISBN of relations of lower-level entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL refers to the primary key attribute of the relation of higher-level entity type BOOK and hence, creates the foreign key constraints. The second approach creates separate relations only for all the lower-level entity types. Each relation of lower-level entity type includes all the attributes of higher-level entity type and its own attributes. This approach is possible when the generalization is disjoint and complete. For example, the relation schemas of the lower-level entity types can be given as TEXTBOOK(ISBN, Book_title, Category, Price, Year, Page_count, Subject) LANGUAGE_BOOK(ISBN, Book_title, Category, Price, Year, Page_count, Language) NOVEL(ISBN, Book_title, Category, Price, Year, Page_count, Type) The second approach has a limitation that the foreign key constraint cannot be created on the relations for lower-level entity type. This is because there is no relation existing to which a foreign key constraint on relations for lower-level entity type can refer. To overcome this limitation, a relation has to be created that contains at least the primary key attribute of the relation for higher-level entity type. 76
Fig. 3.25 The TEXTBOOK, LANGUAGE_BOOK, and NOVEL relations Note that the second approach is not used for overlapping generalization as it may lead to redundancy. For example, if a book is both a textbook as well as a language book, the attributes Book_title, Price, Year, and Page_count have to be stored twice. In addition, the second approach is not used for generalization that is not complete. For example, if a book is neither a textbook nor a language book then another relation, that is, BOOK has to be used to represent such books. An EER diagram containing aggregation expresses relationship between the relationships and a collection of entities/entity types. When EER diagram with aggregation is transformed in the
relation,

the primary key of the relation includes the primary key of the entity type and the primary key of the

relationship type. The descriptive attributes of the entity type are also included in the relation. In addition to the primary key, foreign key constraint can be applied on the relationship types involving aggregation. The primary key of the aggregation refers to the primary keys of the entity type and the relationship type. For example, consider the EER diagram representing aggregation shown in Figure 3.26. Since

the primary key for the relationship type is the union of the primary keys of

the entity types involved,

the primary key for the relationship type WRITES is the union of the primary keys of

the entity type BOOK and AUTHOR. Hence, the schema for the relationship type WRITES can be written as {ISBN,A_ID}

The primary key of the relation, say COPYRIGHT, includes the primary key of the entity type PUBLISHER and the primary key of the

relationship type WRITES. The descriptive attribute COPYRIGHT_ DATE of the relation COPYRIGHT is also included. Now, the schema for the relation COPYRIGHT can be represented as {ISBN,A_ID,P_ID,Copyright_date} and the relation is shown in Figure 3.27. 77
Fig. 3.26 EER diagram with aggregation Fig. 3.27 The COPYRIGHT relation SUMMARY 78
1. A relational database is a collection of relations having distinct names. A relation is used to represent information about an entity and its relationship with other entities in the form of attributes (or columns) and tuples (or rows). 2. The relational model includes three components, namely, structural component (or data structures), manipulative component (or data manipulation), and integrity constraints. 3. Structural component consists of two components, namely, entity types and their relationships, which act as a framework of the given situation. 4. Manipulative component consists of a set of operations that are performed on a table or group of tables. 5. Integrity constraints are a set of rules that governs the entity types and their relationships, and thereby, ensures the integrity of the database. 6. A relation comprises of a relation schema and a relation instance. A relation schema depicts the attributes of the table and a relation instance is a two-dimensional table with a time-varying set of tuples. 7. A relation has certain characteristics. The first is that the tuples in a relation do not have any specified order. The second is that the order in which the values appear in the tuples is relevant. The third is that each tuple in a relation contains a single value for each of its attribute. Finally, each tuple in a relation must be uniquely identified by its contents. 8. The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity
ensures that the data in the database is consistent, accurate, correct, and valid. 9.
Data integrity is of four types, namely, domain integrity, entity integrity, referential integrity, and semantic integrity. 10. Domain integrity can be specified for each attribute by defining its specific range or domain. 11. Entity integrity assigns restriction on the tuples of a relation and ascertains the accuracy and consistency of the database. 12. Referential integrity ensures that the value of
foreign key in the referencing relation must exist in the primary key attribute of the referenced relation, or that
the value of the foreign key is null. 13. Semantic integrity ensures that the data in the database is logically consistent and complete with respect to the real world. 14. Integrity constraints are the rules defined on a relational database schema and satisfied by all the instances of a database in order to model the real-world concepts correctly. 15. The primary key constraint ensures that the attributes, which are declared as primary key must be unique and not null. 16. The unique constraint ensures that a particular set of attributes contains unique values and hence, two tuples cannot have duplicate values in specified attributes. 17. The check constraint can be used to restrict an attribute to have values in a given range, that is, for each tuple in a relation, the values in a certain attribute must satisfy a given condition. One way to achieve this is by applying check constraint during the declaration of the relation. 18. The check constraint can also be applied during domain declaration. It allows specifying a condition that must be satisfied by any value assigned to an attribute whose type is the domain. 19. The not null constraint ensures that the
attribute must not accept null values. 20. A foreign key constraint allows certain attributes in one relation to refer to attributes in another relation. 79

CHAPTER 4 RELATIONAL ALGEBRA AND CALCULUS After reading this chapter, the reader will understand: ? Different query languages used to extract data from the database ? Difference between relational algebra and relational calculus ? Various unary relational algebra operations like select, project, and rename ? Various binary relational algebra operations like set operations, joins, and division operation ? Various set operations like union, intersection, difference, and cartesian product operation ? Different types of joins like equijoin, natural, and outer joins (left, right, and full outer join) ? The aggregate functions that are used to perform queries, which are mathematical in nature ? Two forms of relational calculus, namely, tuple relational calculus and domain relational calculus ? The use of tuple variables for defining queries in tuple relational calculus ? Transformation of the universal and existential quantifiers ? The concept of safe expressions ? The use of domain variables for defining queries in domain
relational calculus ? Expressive power of relational algebra and relational calculus
The two formal languages associated with relational model that are used to specify the basic retrieval requests are relational algebra and relational calculus. Relational algebra provides a foundation for relational model operations and is used as a basis for applying and optimizing queries in relational database management systems. It consists of basic set of operations, which can be used for carrying out basic retrieval operations. Relational calculus, on the other hand, provides declarative notations based on mathematical logic for specifying relational queries. In relational calculus, query is expressed in terms of variables and formulas on these variables. The formula in relational calculus describes the properties of the required resultant. Relational algebra and relational calculus are logically equivalent; however, they differ on the basis of the approach they follow to retrieve data from a relation. Relational algebra uses procedural approach by providing a step-by-step procedure for carrying out a query, whereas relational calculus uses non-procedural approach as it describes the information to be retrieved without specifying the method for obtaining that information. DBMS automatically transforms these non- procedural queries into equivalent procedural queries. This chapter discusses various operations of relational algebra. In addition, it discusses two forms of relational calculus, namely, tuple relational calculus and domain relational calculus. The chapter then concludes with the discussion on the expressive power of

81% **MATCHING BLOCK 80/319** W

relational algebra and relational calculus. 4.1 RELATIONAL ALGEBRA Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input and

result into a new relation as
an output. These operations are divided into two groups. One group consists of operations developed specifically for relational databases such as select, project, rename, join, and division. The other group includes the set-oriented operations such as union, intersection, difference, and cartesian product. Some of the queries, which are mathematical in nature, cannot be expressed using the basic operations of relational algebra. For such queries, additional operations like aggregate functions and grouping are used such as finding sum, average, etc. This section discusses these operations in detail. Things to Remember The relational closure property states that the output generated from any type of relational operation is in the form of relations only. 80
4.1.1 Unary Operations The operations operating on a single relation are known as unary operations. In relational algebra, select, project, and rename are unary operations. Select Operation The select operation retrieves all those tuples from a relation that satisfy a specific condition. It can also be considered as a filter that displays only those tuples that satisfy a given condition. The select operation can be viewed as the horizontal subset of a relation. The Greek letter sigma (σ) is used as a select operator. In general, the select operation is specified as σ &gt;selection_condition&lt; (R) where, selection_condition = the condition on the basis of which subset of tuples is selected R = the name of the relation For example, to retrieve those tuples from BOOK relation whose Category is Novel, the select operation can be specified as σ Category="Novel" (BOOK) The resultant relation of this operation consists of two tuples with Category as Novel and it is shown in Figure 4.1.
The comparison operators (=, ≠, &gt;, ≤, &lt;, ≥) can be used to specify the conditions for selecting required tuples from a relation. Moreover, more than one condition can be concatenated to
one another
to give a more specific condition by using any of the logical operators AND (∧), OR (∨), and NOT (¬). For example, to retrieve those tuples from BOOK relation, whose Category is Language Book and Page_count is greater than 400, the select operation can be specified as σ Category="Language Book" ∧ Page_count&lt;400 (BOOK) Consider another example, to retrieve those tuples from PUBLISHER relation, whose State is either Georgia or Hawaii, the select operation can be specified as σ State="Georgia" ∨ State="Hawaii" (PUBLISHER) Fig. 4.1 Select operation 81

NOTE The comparison operators are used only with the attributes, whose domain is compatible for comparison. The degree (number of attributes) of the relation resulting from a select operation is same as the degree of relation R. The cardinality (number of tuples) of the resulting relation is always less than or equal to the number of tuples in R. The sequence of select operations can be applied in any order, as it is commutative in nature, that is, σ &gt;condition_1&lt; (σ &gt;condition_2&lt; R) = σ &gt;condition_2&lt; (σ &gt;condition_1&lt; R) In other words, the order in which conditions are applied is not significant. Project Operation A relation might consist of a number of attributes, which are not always required. The project operation is used to select some required attributes from a relation while discarding the other attributes. It can be viewed as the vertical subset of a relation. The Greek letter pi (π) is used as project operator. In general, the project operation is specified as π &gt;attribute_list&lt; (R) where, attribute_list = list of required attributes separated by comma R = the name of relation For example, to retrieve only ISBN, Book_title and Price attributes from BOOK relation, the project operation can be specified as π ISBN, Book_title, Price (BOOK) The resultant relation of this operation consists of three attributes of all the tuples as shown in Figure 4.2. Consider another example to retrieve P_ID, State, and Phone attributes from PUBLISHER relation, the project operation can be specified as π P_ID, State, Phone (PUBLISHER) The resultant relation of project operation consists of only those attributes, which are specified in &gt;attribute_list&lt; and are in the same order as they are appearing in the list. Hence, the degree of the resultant relation is equal to the number of attributes in &gt;attribute_list&lt;. If &gt;attribute_list&lt; contains only non-key attributes of relation R, duplicate tuples are likely to appear in the resultant relation. The project operation removes duplicate tuples from the resultant relation. Therefore, the result of the project operation is a valid relation containing a set of unique tuples. This feature of project operation is known as duplicate elimination. For example, consider the project operation given here. 82
Fig. 4.2 Project operation π Category, Page_count (BOOK) The resultant relation of this operation consists of only two attributes, namely, Category and Page_count. Since the project operation removes duplicate tuples from the resultant relation, the tuples (Textbook, 800) and (Novel, 200) appear only once in the resultant relation, although the combination of these values appear repeatedly in BOOK relation as shown in Figure 4.3. 83
Fig. 4.3 Duplicate elimination in project operation Unlike select operation, the commutative property does not hold on project operation, that is, π &gt;list_1&lt; (π &gt;list_2&lt; (R)) ≠ π &gt;list_2&lt; (π &gt;list_1&lt; (R)) In other words, the order in which the list of attributes is specified in project operation is significant. Note that, the select and project operations can also be combined together to create more complex queries. For example, to retrieve P_ID, Address, and Phone attributes from PUBLISHER relation where State is Georgia, the combined operation can be specified as π P_ID, Address, Phone (σ State="Georgia" (PUBLISHER)) 84

This statement first applies select operation on PUBLISHER relation and then applies project operation on the tuples retrieved from select operation. Rename Operation When relational algebra operations are performed, the resultant relations are unnamed, hence cannot be used for later reference. In addition, it might be required to apply more operations on the relation obtained from other operations. In such situations, rename operator proves to be useful. Rename operation is used to provide name to the relation obtained after applying any relational algebra operation. The Greek letter rho (r) is used as a rename operator. Relation obtained from any operation can be renamed by using rename operator as shown here. ρ(R, E) where, ρ = rename operator E = expression representing relational algebra operations R = name given to relation obtained by applying relational algebra operations specified in expression E Consider some examples of using rename operator to rename queries discussed earlier. ρ(R 1 , σ Category="Novel" (BOOK)) ρ(R 2 , π P_ID, State, Phone (PUBLISHER)) ρ(R 3 , π P_ID, Address, Phone (σ State="Georgia" (PUBLISHER))) Here, R 1 , R 2 , and R 3 are the names given to the relations obtained from the respective relational algebra expressions. In these examples, the name of the attributes in new relation is same as in their corresponding original relations. However, attributes in a new relation can also be renamed using the rename operator as shown here. ρ(R(A 1 , A 2 , ...A n), E) where, A 1 , A 2 , ...A n are the new names provided to the attributes of the relation R, obtained after executing relational algebra expression E. For example, consider an expression given here. ρ(R 2 (P 1 , P 2 , P 3), π P_ID, State, Phone (PUBLISHER)) In this example, P 1 , P 2 , P 3 are new attribute names in the relation R 2 for the attributes P_ID, State, and Phone of the relation PUBLISHER, respectively. Sometimes the situation may arise where multiple relational algebra operations need to be applied in a sequence. One way to achieve this is by grouping together the sequence of operations to form a single relational algebra expression. The second way is to apply one operation at a time to create many intermediate result relations. In such situations, rename operator can be used to provide name to the intermediate relations so that they can be referred easily in the subsequent operations. For example, consider a query to retrieve P_ID, Address, and Phone of publishers located at Georgia state. This query can be expressed in a single relational algebra expression as discussed earlier or can be expressed in two steps with the help of rename operator as shown here. ρ(R 1 , σ State="Georgia" (PUBLISHER)) ρ(R 2 , π P_ID, Address, Phone (R 1)) In this example, select operation is applied to retrieve tuples where State is Georgia and resultant relation is named as R 1 . Then, the project operation is applied to retrieve selected attributes, namely, P_ID, Address, and Phone from the intermediate relation R 1 and the resultant relation is named as R 2 . The relation R 2 can further be used for more operations. NOTE The rename operator is optional, it is not always necessary to use it. It can be used as per one's convenience. 85
4.1.2 Binary Operations The operations operating on two relations are known as binary operations. The set operations, join operations, and division operation are all binary operations. Set Operations The standard mathematical operations on sets, namely, union, intersection, difference, and cartesian product are also available in relational algebra. Of these, the three operations, namely, union, intersection, and difference operations require two relations to be union compatible with each other. The two relations are said to be union compatible, if they satisfy these conditions. 1. The degree of both the operand relations must be same. 2. The domain of nth attribute of relation R 1 must be same as that of the nth attribute of relation R 2 . However, the cartesian product can be defined on any two relations, that is, they need not be union compatible. Union Operation The union operation, denoted by ∪, returns a third relation that contains tuples from both or either of the operand relations. Consider two relations R 1 and R 2 , their union is denoted as R 1 ∪ R 2 , which returns a relation containing tuples from both or either of the given relations. The duplicate tuples are automatically removed from the resultant relation. For example, consider the two union compatible relations PUBLISHER_1 and PUBLISHER_2. The union of these two relations consists of tuples from both relations without any duplicate tuples as shown in Figure 4.4. Fig. 4.4 Union operation The union operation is, respectively, commutative and associative in nature, that is, R 1 ∪ R 2 = R 2 ∪ R 1 and R 1 ∪ (R 2 ∪ R 3) = (R 1 ∪ R 2) ∪ R 3 Intersection Operation The intersection operation, denoted by ∩, returns a third relation that contains tuples common to both the operand relations. The intersection of the relations R 1 and R 2 is denoted by R 1 ∩ R 2, which returns a relation containing tuples common to both the given relations. 86
For example, consider two union compatible relations PUBLISHER_1 and PUBLISHER_2. The intersection of these two relations consists of tuples common to both the relations as shown in Figure 4.5. Fig. 4.5 Intersection operation Like, union operation, intersection operation is, respectively, commutative and associative in nature, that is, R 1 ∩ R 2 = R 2 ∩ R 1 and R 1 ∩ (R 2 ∩ R 3) = (R 1 ∩ R 2) ∩ R 3 Learn More Both the union and intersection operations can be performed on any number of relations, as they are associative in nature. Thus, they can be treated as n-ary operations. Difference Operation The difference operation, denoted by – (minus), returns a third relation that contains all
tuples present in one relation, which are not present in

the second relation. The difference of the relations R 1 and R 2 is denoted by R 1 – R 2 . The expression R 1 – R 2 results in a relation containing all tuples from relation R 1 , which are not present in relation R 2 . Similarly, the expression R 2 – R 1 results in a relation containing all tuples from relation R 2 , which are not present in relation R 1 . For example, consider the two union compatible relations PUBLISHER_1 and PUBLISHER_2, the difference of these two relations is shown in Figure 4.6. The difference operation is not commutative in nature, that is, R 1 – R 2 ≠ R 2 – R 1 Note that any relational algebra expression that uses intersection operation can be rewritten by replacing the intersection operation with a pair of difference operation as shown here. R 1 ∩ R 2 = R 1 –(R 1 – R 2) 87
Fig. 4.6 Difference operation Thus, intersection operation is not a fundamental operation. It is easier to write R 1 ∩ R 2 than to write R 1 – (R 1 – R 2). In addition, the intersection operation can also be expressed in terms of difference and union operations as shown here. R 1 ∩ R 2 = (R 1 ∪ R 2) – ((R 1 – R 2) ∪ (R 2 – R 1)) Cartesian Product Operation The cartesian product, also known as cross product or cross join, returns a third relation that contains all possible combinations of the tuples from the two operand relations. The cartesian product is denoted by the symbol ×. The cartesian product of the relations R 1 and R 2 , is denoted by R 1 × R 2 . Each tuple of the first relation is concatenated with all the tuples of the second relation to form the tuples of a new relation. The cartesian product creates tuples with the combined attributes of two relations. Therefore, the degree of a new relation is the sum of the degree of both the operand relations. In addition, the cardinality of the resultant relation is the product of the cardinality of both the operand relations. In other words, if d 1 and d 2 are the degree of the relations R 1 and R 2 , respectively, then the degree of the relation R 1 × R 2 is d 1 + d 2 , and if c 1 and c 2 are the cardinality of relations R 1 and R 2 , respectively, then the cardinality of the relation R 1 × R 2 is c 1 * c 2 . For example, consider two relations AUTHOR_1 with attributes A_ID, Aname, and City, and BOOK_1 with attributes ISBN, Book_title, and Price; the cartesian product of these two relations is shown in Figure 4.7. In this example, each tuple from relation AUTHOR_1 is concatenated with every tuple of relation BOOK_1. The degree of both the operand relations is 3 and hence, the degree of the resultant relation is 6(3+3). The cardinality of the resultant relation is 12, which is the product of the cardinality of the relation AUTHOR_1, that is, 4 and cardinality of the relation BOOK_1, that is, 3. NOTE The cartesian product can be made more useful and meaningful, if it is followed by select and project operations to retrieve meaningful and valuable information from the database. Joins The join operation is one of the most useful and commonly used operations to extract information from two or more relations. A join can be defined as a cartesian product followed by select and project operations. The result of a cartesian product is usually much larger than the result of a join. 88
Fig. 4.7 Cartesian product The join operation joins two relations to form a new relation on the basis of one common attribute present in the two operand relations. That is, if both the operand relations have one common attribute defined over some common domain, then they can be joined over this common attribute. The join results in a new wider relation in which each tuple is formed by concatenating the two tuples, one from each of the operand relations, such that two tuples have the same value for that common attribute. The join is denoted by the symbol . The most general form of the join operation is based on a condition and a pair of operand relations. That is, the join operation is specified as R 1 cond R 2 = σ cond (R 1 × R 2) Note that the condition cond refers to the join condition involving attributes of both relations R 1 and R 2 . There are different types of join operations in relational algebra, namely, equijoin, natural join, outer join, etc. Equijoin A common case of the join operation R 1 R 2 is one in which the join condition consists only of equality condition. This type of join where the only comparison operator used is, = is known as equijoin. The resultant relation of an equijoin always has one or more pairs of attributes that have identical values in every tuple. For example, the relations BOOK and PUBLISHER can be joined for the tuples where BOOK.P_ID is same as PUBLISHER.P_ID. This is an example of equijoin operation and it is specified as shown here. BOOK BOOK.P_ID = PUBLISHER.P_ID PUBLISHER The resultant relation of such a join is shown in Figure 4.8. 89
Fig. 4.8 Equijoin operation In the resultant relation of equijoin operation, the values of the attributes PUBLISHER.P_ID and BOOK.P_ID are same for every tuple as equality condition is specified on these attributes. The attribute publisher ID is appearing with the same name P_ID in both the relations, thus, in order to differentiate these two, the attribute names are qualified by their corresponding relation name using the dot (.) operator. The select operation can be applied on the resultant relation to retrieve only selected tuples. For example, if only those tuples are to be displayed where the Category of book is Textbook and it is published by publishers located in Georgia, then the join operation along with the select operation can be specified as σ Category="Textbook" ∧ State="Georgia" (BOOK BOOK.P_ID = PUBLISHER.P_ID PUBLISHER) The result of this expression consists of a set of tuples from the resultant relation of join operation satisfying the conditions specified. Natural Join The joins having equality condition as their join condition result in two attributes in the resulting relation having exactly the same value. However, if one of the two identical attributes is removed from the result of equijoin, it is known as natural join. The natural join of two relations R 1 and R 2 is obtained by applying a project operation to the equijoin of these two relations in a sequence given here. 1. Find the equijoin of two relations R 1 and R 2 . 90

2. For each attribute A that is common to both relations R 1 and R 2 , project operation is applied to remove the column R 1 .A or R 2 .A. Therefore, if there are m attributes common in both the relations, then m duplicate columns are removed from the resultant relation of equijoin. Learn More Another join similar to natural join is semi-join represented as R 1 R 2 , is a set of all tuples from a relation R 1 having corresponding tuple in the relation R 2 , satisfying the join condition (equality condition). In addition, antijoin represented as R 1 R 2 . is a set of tuples from a relation R 1 not having corresponding tuple in the relation R 2 , satisfying the join condition. The expression to obtain natural join of relations BOOK and PUBLISHER can be specified as ISBN, Book_title, Category, BOOK.P_ID, Pname, Address, Email_ID (BOOK BOOK.P_ID = PUBLISHER.P_ID PUBLISHER) The
resultant relation of such a natural join is shown in Figure 4.9. Fig. 4.9 Natural join operation Note that, the attribute P_ID appears only once in the resultant relation of natural join. The pre-requisite of natural join is that the two attributes involved in the join condition must have the same name. If they have different names then the renaming operation is applied prior to the join operation. Outer join The joins discussed so far are simple and are known as inner joins. An inner join selects only those tuples from both the joining relations that satisfy the joining condition. The outer join, on the other hand, selects all the tuples satisfying the join condition along with the tuples for which no tuples from the other relation satisfy the join condition. An outer join is a type of equijoin that can be used to display all the tuples from one relation, even if there are no corresponding tuples in the second relation, thus preventing information loss. It is commonly used with relations having one-to-many relationship. The three types of outer joins are ? Left outer join: The left outer join includes all the tuples from both the relations satisfying the join condition along with all the tuples in the left relation that do not have a corresponding tuple in the right relation. The tuples from the left relation not satisfying the join condition are concatenated with the tuples having null values for all the attributes from the right relation. The left outer join of relations R 1 and R 2 is specified as R 1 cond R 2 . 91
? Right outer join: The right outer join includes all the tuples from both the relations satisfying the join condition along with all the tuples in the right relation that do not have a corresponding tuple in the left relation. The tuples from the right relation not satisfying the join condition are concatenated with the tuples having null values for all the attributes from the left relation. The right outer join of relations R 1 and R 2 is specified as R 1 cond R 2 . ? Full outer join: A full outer join combines the results of both the left and the right outer joins. The resultant relation contains all records from both the relations, with null values for the missing corresponding tuples on either side. The full outer join of relations R 1 and R 2 is specified as R 1 cond R 2 . Division The division operation, denoted by ÷ is useful for queries of the form 'for all objects having all the specified properties'. To understand the division operation, consider a relation R 1 having exactly two attributes A and B, and a relation R 2 having just one attribute B with the same domain as in R 1 . The division operation R 1 ÷ R 2 is defined as the set of all values of attribute A, such that for every value of attribute B in R 2 , there is a tuple (A, B) in R 1 . In general, for the relations R 1 and R 2 , R 1 ÷ R 2 results in the relation R 3 such that R 3 XR 2 ⊆R 1 . For example, consider the relations R 1 and R 2 with the sample data as shown in Figure 4.10. Fig. 4.10 Division operation In this example, consider the case of division operation on the relations R 1 and R 2 . The relation R 1 ÷ R 2 is the set of values that forms a tuple with every value in the relation R 2 , appearing in a relation R 1 . To elaborate further, the tuples (a 1 , b 2), (a 2 , b 2), (a 3 , b 2) and (a 4 , b 2) are appearing in the relation R 1 where, b 2 is a member of the relation R 2 and (a 1 , a 2 , a 3 , a 4) are members of the relation R 1 ÷ R 2 . Similarly, the results of operations R 1 ÷ R 2 and R 1 ÷ R 4 are shown in Figure 4.10 (b) and 4.10 (c), respectively. Now, take another example of Online Book database, 'retrieve the list of all authors writing books for publishers located in New Jersey and Hawaii'. To achieve this, first the join operation is applied to create a relation R 1 (P_ID, A_ID), which contains the list of publisher ID for whom authors have written the books. Now create a relation R 2 (P_ID) having the list of publisher ID belonging to New Jersey or Hawaii state. The relations R 1 and R 2 can be obtained using these algebraic expressions. ρ (R 1 , π P_ID, A_ID (BOOK BOOK.ISBN=AUTHOR_BOOK.ISBN AUTHOR_BOOK)) ρ (R 2 , π P_ID (σ State="New Jersey" OR State="Hawaii" (PUBLISHER)) The resultant relation of division operation R 1 ÷ R 2 is shown in Figure 4.11. 92
Fig. 4.11 Example of division operation In this example, the relation R 1 ÷R 2 is the set of values that forms a tuple with every value in relation R 2 , which also appears in relation R 1 . To elaborate further, the tuples (P002, A008) and (P003, A008) appear in the relation R 1 where P002 and P003 are from relation R 2 . 4.1.3 Aggregate Functions and Grouping The queries discussed so far are sufficient enough to satisfy most of the requirements of an organization. However, some of the requirements like finding total amount, average price, which are mathematical in nature cannot be expressed by basic relational algebra operations. For such type of queries,

64%	MATCHING BLOCK 83/319	SA	BOOK SIZE.docx (D50092775)

aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as

the result. For example, the aggregate function SUM returns a sum of the collection of numeric values taken as input. Similarly, aggregate functions, namely, AVG, MAX, and MIN are used to find average, maximum, and minimum of the collection of numeric values, respectively. The COUNT function is used for counting tuples or values in a relation. Sometimes, situation may arise where tuples in a relation are required to be grouped, based on the values of an attribute. For example, the tuples of relation BOOK can be divided into groups based on the values of attribute Category. In other words, the books belonging to textbook category form one group, books belonging to novel category form another group, and so on. The aggregate functions can be applied on each group individually, thus, returning a result value for each group separately. For example, for calculating average price for each category of book, AVG function is applied on each category of book separately. In other words, if there are three categories of books, then AVG function is applied to each of the three categories separately, which returns three average values for each category. The grouping of tuples of a relation can be based on one or more attributes. For example, tuples in relation BOOK can be grouped on two attributes, that is, first on the basis of category and then on the basis of publisher ID. The aggregate function operation can be specified using the symbol (called 'script F') as shown here. &gt;attribute_list&lt; &gt;function(attribute)_list&lt; (R) where, &gt;attribute_list&lt; = list of attributes on the basis of which tuples of a relation are to be grouped &gt;function(attribute)_list&lt; = list of functions to be applied on different attributes. For example, consider a relation BOOK, where average, maximum, and minimum price is to be calculated for each category of book separately. The expression to represent this query can be specified as 93
Category AVG(Price), MAX(Price), MIN(Price) (BOOK) The list of attributes on the basis of which tuples are grouped can be omitted from an expression. If grouping attribute is not specified in the expression, then the function is applied individually on all the tuples in the relation. Consider an example where the number of tuples in a relation has to be calculated. The required expression can be specified as COUNT(ISBN) (BOOK) Whenever a function is applied on an attribute, the duplicate values are also included while performing the calculations. However, concatenating the function name with the string _DISTINCT can eliminate duplicate values. For example, consider a relation BOOK where the number of categories is to be calculated. If COUNT function is applied on attribute Category, it counts all the values including duplicate values. However, if COUNT_DISTINCT function is applied, it counts only the unique values. 4.1.4 Example Queries in Relational Algebra Various operations of relational algebra can be combined together to create more advanced queries to extract required data from the database. Some of the queries are discussed here to extract information from Online Book database using operations of relational algebra.
Query 1: Retrieve city, phone and URL of the author whose name is Lewis Ian. π City,Phone,URL (σ Aname="Lewis Ian" (AUTHOR)) In this query, the select operation is used to retrieve all the tuples where Aname is Lewis Ian, and then the project operation is used to display the required attributes, namely, City, Phone and URL for the tuples retrieved.
Query 2: Retrieve name, address, and phone of all the publishers located in New York state. π Pname, Address, Phone (σ State="New York" (
PUBLISHER)) In this query, the select operation is used to retrieve all the tuples where State is New York, and then the project operation is used to display the required attributes, namely, Pname, Address, and Phone for the tuples retrieved.
Query 3: Retrieve title and price of all the textbooks with page count greater than 600. π Book_title, Price (σ Category="Textbook" ∧ Page_count&lt;600 (
BOOK)) In this query, the select operation is used to retrieve all the tuples where Category is Textbook and Page_count is greater than 600 and then the project operation is used to display the required attributes, namely, Book_title and Price for the tuples retrieved.
Query 4: Retrieve ISBN, title, and price of the books belonging to either novel or language book category. π ISBN, Book_title, Price (σ Category="Novel" ∨ Category="Language Book" (
BOOK)) In this query, the select operation is used to retrieve all the tuples where Category is either Textbook or Novel and then the project operation is used to display the required attributes, namely, ISBN, Book_title, and Price for the tuples retrieved. Query 5:
Retrieve ID, name, address, and phone of publishers publishing novels. 94
π P_ID,Pname,Address,Phone (
σ Category="Novel" (BOOK BOOK.P_ID=PUBLISHER.P_ID PUBLISHER)) In this query, the join of relations BOOK and PUBLISHER is created on the basis of common attribute P_ID. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where Category is Novel, and then the project operation is used to display the required attributes, namely, P_ID, Pname, Address, and Phone for the tuples retrieved.
Query 6: Retrieve title and price of all the books published by Hills Publications. π Book_title,Price (
σ Pname="Hills Publications" (BOOK BOOK.P_ID=PUBLISHER.P_ID PUBLISHER)) In this query, the join of relations BOOK and PUBLISHER is created on the basis of common attribute P_ID. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where Pname is Hills Publications, and then the project operation is used to display the required attributes, namely, Book_title and Price for the tuples retrieved.
Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks. π Book_title,R_ID,Rating (

σ Category=“Textbook” (BOOK BOOK.ISBN=REVIEW.ISBN REVIEW)) In this query, the join of relations BOOK and REVIEW is created on the basis of common attribute ISBN. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where Category is Textbook, and then the project operation is used to display the required attributes, namely, Book_title, R_ID, and Rating for the tuples retrieved. Query 8:
Retrieve title, category, and price of all the books written by Charles Smith. π Book_title,Category,Price (
σ Aname=“Charles Smith” ((BOOK BOOK.ISBN=AUTHOR_BOOK.ISBN AUTHOR_BOOK)
AUTHOR_BOOK.A_ID=AUTHOR.A_ID AUTHOR)) In this query, the join of relations BOOK and AUTHOR_BOOK is created on the basis of common attribute ISBN and the resultant relation is joined with the relation AUTHOR on the basis of common attribute A_ID. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where Aname is Charles Smith, and then the project operation is used to display the required attributes, namely, Book_title, Category and Price for the tuples retrieved.
Query 9: Retrieve ID, name, URL of author, and category of the book C ++. π A_ID,Aname,URL,Category (σ Book_title=“C++” ((AUTHOR AUTHOR.A_
ID=AUTHOR_BOOK.A_ID AUTHOR_BOOK) AUTHOR_BOOK.ISBN=BOOK.ISBN BOOK)) In this query, the join of relations AUTHOR and AUTHOR_BOOK is created on the basis of common attribute A_ID and the resultant relation is joined with the relation BOOK on the basis of common attribute ISBN. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where Book_title is C ++, and then the project operation is used to display the required attributes, namely, A_ID, Aname, URL, and Category for the tuples retrieved.
Query 10: Retrieve book title, price, author name, and URL for the publishers Bright Publications. π Book_title,Price,Aname,URL (
σ Pname=“Bright Publications” (((BOOK BOOK.ISBN=AUTHOR_BOOK.ISBN AUTHOR_BOOK)
AUTHOR_BOOK.A_ID=AUTHOR.A_ID AUTHOR) BOOK.P_ID=PUBLISHER.P_ID PUBLISHER)) In this query, four relations are joined on the basis of some common attribute. First the join of the relations BOOK and AUTHOR_BOOK is created based on the common attribute ISBN, then the resultant relation is joined with the relation 95
AUTHOR on the basis of a common attribute A_ID. After this, the resultant relation is joined with the fourth relation PUBLISHER on the basis of a common attribute P_ID. Finally, the select operation is applied on the resultant relation to retrieve all the tuples where Pname is Bright Publications, and then the project operation is used to display the required attributes, namely, Book_title, Price, Aname, and URL for the tuples retrieved. Query 11: Retrieve the name and address of publishers who have not published any books. ρ(R 1 , π P_ID (PUBLISHER)) ρ(R 2 , π P_ID (BOOK)) ρ(R 3 , R 1 – R 2) ρ(R 4 , π Pname, Address (R 3 R3.P_ID=PUBLISHER.P_ID PUBLISHER)) This query uses the difference operation to retrieve ID of publishers who have not published any books. In the first step, the list of P_ID is retrieved from the relation PUBLISHER and stored in the relation R 1 . In the second step, the list of P_ID is retrieved from the relation BOOK and stored in the relation R 2 . The relation R 3 consists of a list of P_ID in the relation R 1 , which is not present in the relation R 2 . In other words, R 3 stores the P_ID of the publishers having a record in relation PUBLISHER who have not published any book; hence, there is no corresponding entry in relation BOOK. Finally, after creating join of relations PUBLISHER and R 3 , Pname and Address for publishers who have not published any books are retrieved in relation R 4 . Query 12: Retrieve the name of all the publishers and the ISBN and title of books published by them (if any). π Pname, ISBN, Book_title (BOOK BOOK.P_ID=PUBLISHER.P_ID PUBLISHER) In this query, the right outer join of relations BOOK and PUBLISHER is created to retrieve the name of all the publishers and the ISBN and title of the books published by them along with those publishers who have not published any book. In that case, the values for attributes ISBN and Book_title will have null values in the resultant relation. Query 13: Retrieve ID and the number of books written by each author. A_ID COUNT(ISBN) (AUTHOR_BOOK) In this query, firstly the tuples of relation AUTHOR_BOOK are grouped on the basis of A_ID and then, the aggregate function COUNT is used to count the number of occurrences of ISBN in each group. Query 14: Retrieve ISBN and the average rating given to each book. ISBN AVG(Rating) (REVIEW) In this query, firstly the tuples of relation REVIEW are grouped on the basis of ISBN and then the aggregate function AVG is used to find the average rating in each group. Query 15: Retrieve the name of the publishers who have published all categories of books. ρ(R 1 , π Pname, Category (BOOK BOOK.P_ID = PUBLISHER.P_ID PUBLISHER)) ρ(R 2 , π Category (BOOK)) ρ(R 3 , R 1 ÷R 2) In this query, firstly the join of relations BOOK and PUBLISHER is created and Pname and Category are retrieved in a relation R 1 . Then the unique values for Category are retrieved from the BOOK relation in another relation R 2 . Finally, the 96
relation R 1 is divided by the relation R 2 to retrieve the name of all those publishers who have published the books of all categories in the relation R 3 . 4.2 RELATIONAL CALCULUS
Relational calculus, an alternative to relational algebra, is a non-procedural or declarative query
language
as it
specifies what is to be retrieved rather than how to retrieve it.

In relational calculus, queries are expressed in the form of variables and formulas consisting of these variables. The formula specifies the properties of the resultant relation without giving specific procedure for evaluating it. There are two forms of relational calculus, namely, tuple relational calculus (TRC) and domain relational calculus (DRC). In tuple relational calculus, the variable ranges over tuples from a specified relation and in domain relational calculus, the variable ranges over attributes from a specified relation. The Structured Query Language (SQL) is influenced by TRC; however, the Query-By-Example (QBE) is influenced by DRC. 4.2.1

100%	MATCHING BLOCK 84/319	SA	BOOK SIZE.docx (D50092775)

Tuple Relational Calculus The tuple relational calculus is based on

tuple variables, which takes tuples of a specific relation as its values.

80%	MATCHING BLOCK 85/319	SA	BOOK SIZE.docx (D50092775)

A simple tuple relational calculus query is of the form {T|P(T)} where, T

is a tuple variable and P(T) is a condition or formula for retrieving required tuples from a relation. The resultant relation of this query

90%	MATCHING BLOCK 86/319	SA	DCAP 011.docx (D142453284)

is the set of all tuples T for which the condition P(T) evaluates to true.

For example, the query to retrieve all books having price greater than $30 can be specified in the form of tuple calculus expression as {T | BOOK(T) ∧ T.Price&lt;30} The condition BOOK(T) specifies that the tuple variable T is defined for the relation BOOK. The query retrieves the set of all the tuples T satisfying the condition T.Price&lt;30. Note that T.Price refers to the attribute Price of tuple variable T. This query extracts all the attributes of the relation BOOK. However, if only selected attributes have to be retrieved, then the attribute list can be specified in the tuple calculus expression as {T.ISBN, T.Book_title, T.Price | BOOK(T) ∧ T.Price&lt;30} This expression extracts only three attributes, namely, ISBN, Book_title, and Price of the tuple set satisfying the given condition. The tuple calculus expression consists basically of three components, which are ? The relation R for which tuple variable T is defined ? A condition P(T) on the basis of which set of tuples is to be retrieved. ? An attribute list specifying the required attributes to be retrieved for the tuples satisfying the given condition. Expressions and Formulas in Tuple Relational Calculus A general expression of the tuple relational calculus consists of a tuple variable T and a formula P(T). A formula can consist of more than one tuple variable. A formula is made up of atoms and atoms can be of any of the forms given here. 1. R(T), where T is a tuple variable and R is a relation. 2. T1.A1 opr T2.A2, where opr is a comparison operator (=, ≠, &gt;, ≤, &lt;, ≥), T 1 and T 2 are tuple variables. A 1 is an attribute of the relation on which T 1 ranges and A 2 is an attribute of the relation on which T 2 ranges. 3. T.A opr c or c opr T.A, where opr is any comparison operator, T is a tuple variable, A is an attribute of the relation on which T ranges, and c is a constant in the domain of attribute A.

87%	MATCHING BLOCK 88/319	SA	BOOK SIZE.docx (D50092775)

Each of the atoms evaluates to either true or false for a specific combination of tuples

known as the truth value of an atom. A formula is built from one or more atoms concatenated with the help of the logical operators (¬, ∧,∨)

61%	MATCHING BLOCK 87/319	W

by using these rules 97 1. An atom is a formula. 2. If F is a formula, then so is ¬F. 3. If F 1 and F 2 are formulae, then so are

F 1 ∧ F 2 , F 1 ∨ F 2 and F 1 ⇒ F 2 , where F 1 ⇒ F 2 is also equivalent to (¬(F 1) ∨ F 2). 4. If F is a formula, then so is ∃T(F), where T is a tuple variable. 5. If F is a formula, then so is ∀T(F), where T is a tuple variable. Note that in the last two clauses, special symbols called quantifiers, namely, existential quantifier (∃) and universal quantifier (∀) are used to quantify the tuple variable T. The expression, ∀T, means 'for every occurrence of T' and the expression ∃T, means 'for some occurrence of T'. A tuple variable T is said to be bound, if it is quantified, that is,

96%	MATCHING BLOCK 89/319	SA	BOOK SIZE.docx (D50092775)

it appears in (∃T) or (∀T) clause, otherwise, it is free. A tuple variable

T can be said to be free or bound in a formula, on the basis of the rules given here. ? A tuple variable T is free

71%	MATCHING BLOCK 90/319	SA	BOOK SIZE.docx (D50092775)

in a formula F, which is an atom. ? A tuple variable T is free or bound in a formula of the forms (F 1 ∧ F 2), (F 1 ∨ F 2) and ¬F depending on whether it is free or bound in F 1 or F 2 (

if it occurs in either of the formulas). Note that in a formula of the form F=(F 1 ∨ F 2), a tuple variable may be free in F 1 and bound in F 2 , or vice versa. In such cases, one occurrence of the tuple variable is bound and the other

91%	MATCHING BLOCK 91/319	SA	BOOK SIZE.docx (D50092775)

is free in F. ? All free occurrences of a tuple variable T in F are bound in a formula F′ of the form F′ = (∃T) (F) or F′ =(∀T) (F). The tuple variable is bound to the quantifier specified in F′.

A query can be evaluated on the basis of a given instance of the database. The truth value of a formula can be derived as given here. 1. An atom (formula) is true if tuple variable T is assigned a tuple from a relation R, otherwise it is false. 2. ¬F is true if F is false, and it is false if F is true. 3. F 1 ∧ F 2 is true if both F 1 and F 2 are true, otherwise it is false. 4. F 1 ∨ F 2 is false if both F 1 and F 2 are false, otherwise it is true. 5. In F 1 ⇒ F 2 , F 2 is true whenever F 1 is true, otherwise it is false. 6. (∃T)(F) is true if F is true for some (at least

85%	MATCHING BLOCK 92/319	SA	BOOK SIZE.docx (D50092775)

one) tuple assigned to free occurrences of T in F, otherwise it is false. 7. (∀

T)(F) is true if F is

66%	MATCHING BLOCK 93/319	SA	BOOK SIZE.docx (D50092775)

true for every tuple (universal) assigned to free occurrences of T in F, otherwise it is false. Transforming the Universal and

Existential Quantifiers An expression containing a universal quantifier can be transformed into an equivalent expression containing an existential quantifier and vice versa. The steps to perform this type of transformation are 1. Replace one type of quantifier into the other and precede the quantifier by NOT (¬) operator. 2. Replace AND(∧) with OR(∨) operator and vice versa. 3. Formula is preceded by NOT (¬) operator, as a result negated formula becomes affirmed and vice versa. Things to Remember In a formula, when multiple tuple variables are quantified, the evaluation takes place from inside to outside. For example, in the formula ∃T∀S(F), first ∀S will evaluate and then ∃T will be evaluated. The sequence of identical quantifiers can be altered with no other type of quantifier appearing in between. Consider some examples of this transformation given here. (∃T)(F) ≡ ¬(∀T)(¬F) (∀T)(F) ≡ ¬(∃T)(¬F) (∃T)(F 1 ∧F 2) ≡ ¬(∀T)(¬F 1 ∨¬F 2) (∀T)(F 1 ∨F 2) ≡ ¬(∃T)(¬F 1 ∧¬F 2) (∃T)(¬F 1 ∨¬F 2) ≡ ¬(∀T)(F 1 ∧F 2) (∀T)(¬F 1 ∧¬F 2) ≡ ¬(∃T)(F 1 ∨F 2) 98
Note that the symbol ≡ represents equivalent to. Example Queries in Tuple Relational Calculus The queries expressed in relational algebra can also be expressed in tuple relational calculus. The difference is only in the notation used to express the queries. The queries discussed in relational algebra can be expressed in tuple relational calculus
as shown here. Query 1: Retrieve city, phone, and URL of the author whose name is Lewis Ian. {T.City, T.Phone, T.URL | AUTHOR(T) ∧ T.Aname="Lewis Ian"}

In this query, tuple variable T for the relation AUTHOR and the condition to retrieve required tuples from this relation is specified after the bar (|). The tuples satisfying the given condition are assigned to T, and the attribute values for City, Phone, and URL are displayed for the tuples retrieved.
Query 2: Retrieve name, address, and phone of all the publishers located in New York state. {T.Pname, T.Address, T.Phone | PUBLISHER(T) ∧ T.State="New York"}
In this query, the tuple variable T for the relation PUBLISHER is defined, and the tuples satisfying the given condition are assigned to T. Then, the attribute values for Pname, Address, and Phone are displayed for the tuples retrieved.
Query 3: Retrieve title and price of all the textbooks with page count greater than 600. {T.Book_title, T.Price | BOOK(T) ∧ T.Category="Textbook" ∧ T.Page_count&lt;600}
In this query, the tuple variable T for the relation BOOK is defined, and the tuples satisfying the given condition are assigned to T. Then, the attribute values for Book_title and Price are displayed for the tuples retrieved.
Query 4: Retrieve ISBN, title and price of the books belonging to either novel or language book category. {T.ISBN, T.Book_title, T.Price | BOOK(T) ∧ (T.Category="Novel" ∨ T.Category="Language Book")} In this query, the tuple variable T for the relation BOOK is defined, and the tuples satisfying the given condition are assigned to T. Then, the attribute values for ISBN, Book_title, and Price are displayed for the tuples retrieved. Query 5: Retrieve ID, name, address, and phone of publishers publishing novels. {T.P_ID, T.Pname, T.Address, T.Phone | PUBLISHER(T) ∧ (∃S)(BOOK(S) ∧ S.P_ID=T.P_ID ∧ S.Category="Novel")} In this query, tuple variables T and S for the relations PUBLISHER and BOOK, respectively, are defined. Of these, T is a free tuple variable and S is bound to the existential quantifier. Note that the condition S.P_ID=T.P_ID is a join condition for PUBLISHER and BOOK relations. The attribute values of P_ID, Pname, Address, and Phone of all the tuples satisfying the specified condition are displayed.
Query 6: Retrieve title and price of all the books published by Hills Publications. {T.Book_title, T.Price | BOOK(T) ∧ (∃S)(PUBLISHERS(S) ∧ S.P_ID =T.P_ID ∧ S.Pname = "Hills Publications")}
In this query, the tuple variables T and S for the relations BOOK and PUBLISHER, respectively, are defined. Here, T is a free tuple variable and S is bound to the existential quantifier. The two relations are joined on the basis of common attribute P_ID. The attribute values of Book_title and Price of all the tuples satisfying the specified condition are displayed.
Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks. {T.Book_title,
S.R_ID, S.Rating | BOOK(T) ∧ REVIEW(S) ∧ T.Category="Textbook" ∧ T.ISBN = S.ISBN} 99
In this query, the tuple variables T and S for the relations BOOK and REVIEW, respectively, are defined. The two relations are joined on the basis of common attribute ISBN. The attribute values of Book_title, R_ID, and Rating of all the tuples satisfying the specified condition are displayed. Query 8:
Retrieve title, category, and price of all the books written by Charles Smith. {
T.Book_title, T.Category, T.Price | BOOK(T) ∧ ((∃S)(∃P)(AUTHOR(S) ∧ AUTHOR_BOOK(P) ∧ S.Aname="Charles Smith" ∧ T.ISBN=P.ISBN ∧ P.A_ID=S.A_ID))} In this query, the tuple variables T, S, and P for the relations BOOK, AUTHOR, and AUTHOR_BOOK, respectively, are defined. Here, T is a free tuple variable, whereas S and P are bound to the existential quantifiers. The relations BOOK and AUTHOR_BOOK are joined on the basis of common attribute ISBN and the resultant relation is joined with the relation AUTHOR on the basis of common attribute A_ID. After this, the attribute values for Book_title, Category, and Price of all the tuples where Aname is Charles Smith, are displayed.
Query 9: Retrieve ID, name, URL of author, and category of the book C ++. {T.A_ID,
T.Aname, T.URL, S.Category | AUTHOR(T) ∧ BOOK(S) ∧ S.Book_title ="C++" ∧ ((∃P)(AUTHOR_BOOK(P) ∧ T.A_ID=P.A_ID ∧ P.ISBN=S.ISBN))} In this query, the tuple variables T, S, and P for the relations AUTHOR, BOOK, and AUTHOR_BOOK, respectively, are defined. Here, T and S are free tuple variables whereas P is bound to the existential quantifier. The relations AUTHOR and AUTHOR_BOOK are joined on the basis of common attribute A_ID, and the resultant relation is joined with the relation BOOK on the basis of common attribute ISBN. After this, the attribute values for A_ID, Aname, URL, and Category of all the tuples where Book_title is C ++ are displayed.
Query 10: Retrieve book title, price, author name, and URL for the publishers Bright Publications. {T.Book_title, T.Price, S.Aname, S.URL | BOOK(T) ∧ AUTHOR(S) ∧ ((∃P) (∃R)(PUBLISHER(P) ∧ AUTHOR_BOOK(R) ∧ T.ISBN=R.ISBN ∧ R.A_ID=

S.A_ID ∧ T.P_ID=P.P_ID ∧ P.Pname="Bright Publications"))} In this query, the tuple variables T, S, P, and R for the relations BOOK, AUTHOR, PUBLISHER, and AUTHOR_BOOK, respectively, are defined. Among these, T and S are free tuple variables, whereas P and R are bound to existential quantifiers. Here, four relations are joined on the basis of some common attribute. First the join of the relations BOOK and AUTHOR_BOOK is created on the basis of the attribute ISBN, then the resultant relation is joined with the relation AUTHOR on the basis of the attribute A_ID. After this, the resultant relation is joined with the fourth relation PUBLISHER on the basis of the attribute P_ID. Finally, the attribute values for Book_title, Price, Aname, and URL of all tuples where Pname is Bright Publications, are displayed. Query 11: Retrieve name and address of publishers who have not published any books. {T.Pname, T.Address | PUBLISHER(T) ∧ (¬(∃S)(BOOK(S) ∧ T.P_ID=S.P_ID))} In this query, the tuple variables T and S for the relations PUBLISHER and BOOK, respectively, are defined. Here, T is a free tuple variable, whereas S is bound to existential quantifier. Note that the negation operator (¬) is used to retrieve the tuples, which do not satisfy the join condition, that is, it retrieves the attributes Pname and Address of the publishers who have no corresponding books in relation BOOK. This query can also be expressed by using universal quantifier instead of existential quantifier by following general transformation rules as shown here. {T.Pname, T.Address | PUBLISHER(T) ∧ ((∀S)(¬(BOOK(S)) ∨ ¬(T.P_ID=S.P_ID)))} Safe Expressions An expression of tuple relational calculus containing universal quantifier, existential quantifier or negation condition may lead to a relation of infinite number of tuples. For example, consider an expression given here. 100
{T | ¬BOOK(T)} This expression appears to be syntactically correct but it refers to all the tuples in the universe not belonging to BOOK relation, which are infinite in number. These types of expressions are known as unsafe expressions. Therefore, care must be taken while using quantifier symbols and negation operator in a relational calculus expression to ensure safe expressions. A safe expression is an expression, which results in a relation with finite number of tuples. Safe expressions can be defined

73%	MATCHING BLOCK 94/319	SA	BOOK SIZE.docx (D50092775)

more precisely by introducing the concept of the domain of a tuple relational calculus expression, E. The domain of

expression E, denoted by Dom(E),

73%	MATCHING BLOCK 95/319	SA	BOOK SIZE.docx (D50092775)

is the set of all values appearing either as constant values in the expression E, or appearing in any of the

tuples of relations referred by E. In other words, the domain of E

70%	MATCHING BLOCK 96/319	SA	BOOK SIZE.docx (D50092775)

is the set of all values that appear as constant in E, or appear in

one or more relations whose names appear in E. Therefore, the domain of the above expression includes all the values appearing in the relation BOOK. Similarly, the domain for the Query 5 includes all the values appearing in the relations BOOK and PUBLISHER. Hence, an expression E, is said to be safe if all the values appearing in the resultant relation are from the domain of E, Dom(E). The expression {T | ¬BOOK(T)} is said to be safe if all the values appearing in the resultant relation are from the domain, Dom(BOOK). However, this expression is unsafe as the resultant relation of this expression includes all the tuples in the universe not appearing in the BOOK relation, hence not belonging to the domain Dom(BOOK). All other expressions discussed in tuple relational calculus are safe expressions. 4.2.2 The Domain Relational Calculus The domain relational calculus is based on domain variables, which unlike tuple relational calculus, range over the values from the domain of an attribute, rather than values for an entire tuple.

70%	MATCHING BLOCK 100/319	SA	BOOK SIZE.docx (D50092775)

A simple domain relational calculus query is of the form {T | P(T)} where, T

represents a set of domain variables (x 1 , x 2 , ..., x n) that ranges over domains of attributes (A 1 , A 2 , ..., A n) and P represents the formula on T. Like, tuple relational calculus, formulae in domain relational calculus are built up from atoms and an atom can be of any of the forms given here. 1. R(

	W
	MATCHING BLOCK 97/319

x 1 , x 2 , ..., x n), where R is a relation with n attributes and x 1 , x 2 , ..., x n are domain variables or domain constants

for the corresponding n attributes. 2. x 1 opr x 2 , where opr is a comparison operator (=, ≠, &gt;, ≤, &lt;, ≥) and x 1 and x 2 are domain variables. Note that the domain of attributes x 1 and x 2 are compatible for comparison. 3. x opr c or c opr x, where opr is a comparison operator,

67%	**MATCHING BLOCK 98/319**	W

x is a domain variable and c is a constant in the domain of the attribute for which x is a domain variable. Formulas are built from atoms by using these rules 1. An atom is a formula. 2. If F is a formula, then so is ¬F. 3. If F 1 and F 2 are formulae, then so are

F 1 ∧ F 2 , F 1 ∨ F 2 and F 1 ⇒ F 2 . 4. If F is a formula, then so is ∃x(F), where x is a domain variable. 5. If F is a formula, then so is ∀x(F), where x is a domain variable. Like in tuple relational calculus,

87%	**MATCHING BLOCK 104/319**	SA	BOOK SIZE.docx (D50092775)

atoms evaluate to either true or false for a specific set of

values. In rule 1, an atom (formula) is true, if the domain variable is assigned values corresponding to a tuple of the specified relation R. Similarly, in the remaining rules, if the domain variables are assigned values that satisfy the condition, then the atom is true, otherwise it is false. Example Queries in Domain Relational Calculus The queries expressed in relational algebra or tuple relational calculus can also be expressed in domain relational calculus. The difference is only in the notation used to express the queries. In domain relational calculus, the queries based on the 101
relation AUTHOR require seven domain variables a 1 , a 2 , ..., a 7 to range over the domain of each attribute of the relation in order. Similarly, for the relations BOOK, PUBLISHER, AUTHOR_BOOK, and REVIEW, the number of domain variables required is eight (b 1 , b 2 , ..., b 8), six (p 1 , p 2 , ..., p 6), two (c 1 , c 2) and three (r 1 , r 2 , r 3), respectively. In domain relational calculus, the domain variables for the attributes to be displayed are specified before the bar (|) and the relations and conditions to retrieve the required tuples are specified after the bar (|). The queries discussed earlier can be expressed in domain relational calculus
as shown here. Query 1: Retrieve city, phone, and URL of the author whose name is Lewis Ian. {

88%	**MATCHING BLOCK 99/319**	W

a 4 , a 6 , a 7 | (∃a 2) (AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 , a 6 , a 7) ∧ a 2 =“

Lewis Ian”)} The required attributes City, Phone, and URL to be displayed are specified by free domain variables a 4 , a 6 , and a 7 , respectively, for the relation AUTHOR. Existential quantifier quantifies the variable a 2 for which condition is specified.
Query 2: Retrieve name, address, and phone of all the publishers located in New York state. {
p 2 , p 3 , p 5 | (∃p 4) (PUBLISHER(p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ p 4 =“New York”)} The required attributes Pname, Address, and Phone to be displayed are specified by free domain variables p 2 , p 3 , and p 5 , respectively, for the relation PUBLISHER. Existential quantifier quantifies the variable p 4 for which condition is specified.
Query 3: Retrieve title and price of all the textbooks with page count greater than 600. {

88%	**MATCHING BLOCK 101/319**	W

b 2 , b 4 | (∃b 3) (∃b 7) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧

b 3 ="Textbook" ∧ b 7 &lt;600)} The required attributes Book_title and Price to be displayed are specified by free domain variables b 2 and b 4 , respectively, for the relation BOOK. Existential quantifiers quantify the variables b 3 and b 7 for which conditions are specified.
Query 4: Retrieve ISBN, title and price of the books belonging to either novel or language book category. {

b 1 , b 2 , b 4 | (∃b 3) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ (

b 3 ="Novel" ∨ b 3 ="Language Book"))} The required attributes ISBN, Book_title, and Price to be displayed are specified by free domain variables b 1 , b 2 , and b 4 , respectively, for the relation BOOK. Existential quantifier quantifies the variable b 3 for which conditions are specified. Query 5: Retrieve ID, name, address, and phone of publishers publishing novels. {p 1 , p 2 , p 3 , p 5 | (∃b 3) (∃b 8) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ PUBLISHER(p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ b 8 = p 1 ∧ b 3 ="Novel")} The required attributes P_ID, Pname, Address, and Phone to be displayed are specified by free domain variables p 1 , p 2 , p 3 , and p 5 , respectively, for the relation PUBLISHER. Existential quantifiers quantify the variables b 3 and b 8 for which conditions are specified. The variable p 1 appearing in a condition is not quantified as it is appearing as a free domain variable before the bar (|). Note that the condition b 8 = p 1 is a join condition as it relates two domain variables ranging over common attributes from two different relations. This condition creates join of two relations BOOK and PUBLISHER.
Query 6: Retrieve title and price of all the books published by Hills Publications. {
b 2 , b 4 | (∃p 2) (∃p 1) (∃b 8) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ PUBLISHER (p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ p 1 =b 8 ∧ p 2 ="Hills Publications")} The required attributes Book_title and Price to be displayed are specified by free domain variables b 2 and b 4 , respectively, for the relation BOOK. Existential quantifiers quantify the variables p 2 , p 1 , and b 8 for which conditions are specified. Note that the condition p 1 = b 8 is a join condition as it creates a join of two relations PUBLISHER and BOOK.
Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks. 102
{
b 2 , r 1 , r 3 | (∃b 3) (∃b 1) (∃r 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ REVIEW(r 1 , r 2 , r 3)∧ b 1 =r 2 ∧ b 3 ="Textbook")} The required attributes Book_title, R_ID, and Rating to be displayed are specified by free domain variables b 2 for the relation BOOK, r 1 and r 3 for the relation REVIEW, respectively. Existential quantifiers quantify the variables b 3 , b 1 , and r 2 for which conditions are specified. Note that, the condition b 1 = r 2 is a join condition as it creates a join of two relations BOOK and REVIEW. Query 8:
Retrieve title, category, and price of all the books written by Charles Smith. {
b 2 ,
b 3 , b 4 | (∃b 1) (∃

c 2) (∃c 1) (∃a 1) (∃a 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 , a 6 , a 7) ∧ AUTHOR_BOOK(c 1 , c 2) ∧ b 1 = c 2 ∧ c 1 = a 1 ∧ a 2 ="

Charles Smith")} The required attributes Book_title, Category, and Price to be displayed are specified by free domain variables b 2 , b 3 , and b 4 , respectively, for the relation BOOK. Existential quantifiers quantify the variables b 1 , c 2 , c 1 , a 1 , and a 2 for which conditions are specified. The conditions b 1 =c 2 and c 1 =a 1 creates a join of relations BOOK, AUTHOR_BOOK, and AUTHOR.
Query 9: Retrieve ID, name, URL of author, and category of the book C++. {

a 1 , a 2 , a 7 , b 3 | (∃c 1) (∃b 1) (∃c 2) (∃b 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ 2 ="

C++")} The required attributes A_ID, Aname, URL, and Category to be displayed are specified by free domain variables a 1 , a 2 , a 7 for the relation AUTHOR and b 3 for the relation BOOK, respectively. Existential quantifiers quantify the variables c 1 , b 1 , c 2 , and b 2 for which conditions are specified. Note that variable a 1 appearing in a condition is not quantified as it is appearing as free domain variable before the bar (|). The conditions a 1 = c 1 and c 2 = b 1 creates a join of relations AUTHOR, AUTHOR_BOOK, and BOOK. Query 10: Retrieve book title, price, author name, and URL for the publishers Bright Publications. {

b 2 , b 4 , a 2 , a 7 | (∃ a1) (∃c 1) (∃c 2) (∃b 1) (∃b 8) (∃p 1) (∃p 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8 ∧ AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 ,

a 6 , a 7) ∧
AUTHOR_BOOK (c 1 , c 2) ∧ PUBLISHER(p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ a 1 =c 1 ∧ c 2 =b 1 ∧ b 8 =p 1 ∧ p 2 = "Bright Publications")} The required attributes Book_title, Price, Aname, and URL to be displayed are specified by free domain variables b 2 , b 4 for the relation BOOK and a 2 , a 7 for the relation AUTHOR, respectively. Existential quantifiers quantify the variables a 1 , c 1 , c 2 , b 1 , b 8 , p 1 , and p 2 for which conditions are specified. The conditions a 1 =c 1 , c 2 =b 1 , and b 8 = p 1 creates a join of relations AUTHOR, AUTHOR_BOOK, BOOK, and PUBLISHER. Query 11: Retrieve name and address of publishers who have not published any books. {p 2 , p 3 | (∃p 1) (PUBLISHER(p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ (¬(∃b 8) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ p 1 =b 8)))} The required attributes Pname and Address to be displayed are specified by free domain variables p 2 , and p 3 , respectively for the relation PUBLISHER. Existential quantifiers quantify the variables p 1 and b 8 for which conditions are specified. The negation operator (¬) is used to retrieve those publisher IDs who have not published any book. This query can also be expressed by using universal quantifier instead of existential quantifier by following general transformation rules as shown here. {p 2 , p 3 | (∃p 1) (PUBLISHER(p 1 , p 2 , p 3 , p 4 , p 5 , p 6) ∧ ((∀b 7) (¬(BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8)) ∨ ¬(p 1 =b 7))))} 4.3 EXPRESSIVE POWER OF RELATIONAL ALGEBRA AND RELATIONAL CALCULUS As discussed earlier, the relational algebra provides a procedure for solving a problem, whereas relational calculus simply describes what the requirement is. However, relational algebra and relational calculus are logically equivalent. That is, for every safe relational calculus expression there exists equivalent relational algebra expression and vice versa. In other words, there is one-to-one mapping between the two and the difference lies only in the way the query is expressed. 103

The expressive power of relational algebra is frequently used as a metric

to measure the power of any relational database query language. A query language is said to be relationally complete if it is at least as powerful as relational algebra, that is, if it can express any query expressed in relational algebra. Relational completeness has become an important basis for comparing the expressive power of various relational database query languages. Most of the relational database query languages are relationally complete. In addition, they have more expressive power than relational algebra or relational calculus with the inclusion of various advanced operations such as aggregate functions, grouping, and ordering. SUMMARY 1.

Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input, and

result into a new relation as
an output. 2. Relational algebra operations are divided into two groups. One group consists of operations developed specifically for relational databases such as select, project, rename, join, and division. The other group includes the set-oriented operations such as union, intersection, difference, and cartesian product. 3. The select operation retrieves all those tuples from a relation that satisfy a specific condition. The Greek letter sigma (σ) is used as a select operator. 4. The project operation is used to select some required attributes from a relation while discarding the other attributes. The Greek letter pi (π) is used as project operator. 5. Rename operation is used to provide name to the relation obtained after applying any relational algebra operation. The Greek letter rho (ρ) is used as a rename operator. 6. The union operation, denoted by ∪, returns a third relation that contains tuples from both or either of the operand relations. 7. The intersection operation, denoted by ∩, returns a third relation that contains tuples common to both the operand relations. 8. The difference operation, denoted by – (minus), returns a third relation that contains all
tuples present in one relation, which are not present in

the second relation. 9. The cartesian product, also known as cross product or cross join, returns a third relation that contains all possible combinations of the tuples from the two operand relations. The cartesian product is denoted by the symbol ×. 10. A join can be defined as a cartesian product followed by select and project operations. The join operation joins two relations to form a new relation on the basis of one common attribute present in the two operand relations. 11. A common case of the join operation in which the join condition consists only of equality condition is known as equijoin. 12. The equijoin operation results in two attributes in the resulting relation having exactly the same value. If one of the two identical attributes is removed from the result of equijoin, it is known as natural join. 13. An inner join selects only those tuples from both the joining relations that satisfy the joining condition. The outer join, on the other hand, selects all the tuples satisfying the join condition along with the tuples for which no tuples from the other relation satisfy the join condition. 14.

69%	MATCHING BLOCK 109/319	W

There are three types of outer joins, namely, left outer join, right outer join, and full outer join. 15.

Some of the requirements like finding total amount, average price, which are mathematical in nature, cannot be expressed by basic relational algebra operations. For such type of queries,

64%	MATCHING BLOCK 111/319	SA	BOOK SIZE.docx (D50092775)

aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as

the result. 16.
Relational calculus, an alternative to relational algebra, is a non-procedural or declarative query
language
as it
specifies what is to be retrieved rather than how to retrieve it.
In relational calculus, queries are expressed in the form of variables and formulas consisting of these variables. 17. There are two forms of relational calculus, namely, tuple relational calculus (TRC) and domain relational calculus (DRC). 18. The tuple relational calculus is based on tuple variables, which takes tuples of a specific relation as its values. 19. An expression of tuple relational calculus containing universal quantifier, existential quantifier or negation condition may lead to a relation of infinite number of tuples. These types of expressions are known as unsafe expressions. A safe expression, on the other hand, results in a relation with finite number of tuples. 104
20. The domain relational calculus is based on domain variables, which unlike tuple relational calculus, range over the values from the domain of an attribute, rather than values for an entire tuple. 21.

87%	MATCHING BLOCK 113/319	SA	MCSDSC-2.2 Relational database Management syst ... (D151484310)

The expressive power of relational algebra is frequently used as a metric

to measure the power of any relational database query language. A query language is said to be relationally complete if it is at least as powerful as relational algebra, that is, if it can express any query expressed in relational algebra. 105

CHAPTER 5 STRUCTURED QUERY LANGUAGE After reading this chapter, the reader will understand: ? The basic features of SQL ? The concept of schema and catalog in SQL ? Different types of data types available in SQL ? DDL commands to create relations in SQL ? Various constraints that can be applied on the attributes of a relation ? DDL commands to alter the structure of a relation ? DML commands to manipulate the data stored in the database ? Various clauses of SELECT command that can be used to extract data from database ? Various set operations like union, intersect, and minus ? The use of INSERT, UPDATE, and DELETE commands to insert, update, and delete tuples in a relation ? Searching of null values in a relation ? The use of various aggregate function in select query ? The use of GROUP BY and HAVING clause for the grouping of tuples in a relation ? Combining tuples from two relations using join queries ? The concept of nested queries ? Specifying complex constraints using the concept of assertions ? The concept views, whose contents are derived from already existing relations and does not exist in physical form ? The concept of stored procedures and functions in SQL ? The use of trigger for automatic execution of the stored procedures ? The difference between embedded and dynamic SQL ? The use of cursor for extracting multiple tuples from a relation ? The role of Open Database Connectivity (ODBC) for establishing connection with databases ? The use of Java Database Connectivity (JDBC) for establishing connectivity with databases in Java ? The use of SQL-Java standard for embedding SQL statements in Java program The structured query language (SQL) pronounced as “ess-que-el”, is a language which can be used for retrieval and management of data stored in relational database. It is a non-procedural language as it specifies what is to be retrieved rather than how to retrieve it.
It can be used for defining the structure of data, modifying data in the database, and specifying the security constraints. SQL is an interactive query language that helps users to execute complicated queries in minutes, sometimes even in seconds as compared to the number of days taken by a programmer to write a program for those complicated queries. In addition, commands in SQL resemble simple English statements, making it easier to learn and understand. These features make SQL as one of the major reasons of the success of relational databases.
Some common relational database management systems that use SQL are Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.
Donald D. Chamberlin and Raymond F. Boyce developed the first version of SQL, also known as
Sequel as part of the System R project in the early 1970s
at IBM. Later on, SQL was adopted as a standard by the ANSI in 1986, and was subsequently adopted as an ISO standard in 1987. This version of SQL was called SQL-86. In 1989, a revised standard commonly known as SQL-89 was published. The demand for more advanced SQL version led to the development of new version of SQL called SQL-92. This version addressed several weaknesses in SQL-89 standard. In 1999, the standard SQL-99 was released by ANSI/ISO. This version addresses some of the advanced areas of modern SQL systems, such as object- relational database concepts, integrity management, etc. The complete coverage of SQL cannot be provided in a single chapter hence; only the fundamental concepts have been discussed in this chapter. Moreover, the implementation of various SQL commands may differ from version to version. 106

5.1 BASIC FEATURES OF SQL SQL has proved to be the standard language as same set of commands are used to create, retrieve, and modify data regardless of the system they are working on. SQL provides different types of commands, which can be used for different purposes. These commands are divided into two major categories. ? Data definition language (DDL): DDL provides commands that can be used to create, modify, and delete database objects. The database objects include relation schemas, relations, views, sequences, catalogs, indexes, etc. ? Data manipulation language (DML): DML provides commands that can be used to access and manipulate the data, that is, to retrieve, insert, delete, and update data in a database. Learn More The first commercially available implementation of SQL was released in 1979 by Relational Software Inc., which is known as Oracle Corporation today. Thus, Oracle is the pioneer RDBMS that started using SQL. In addition, there are various techniques like embedded SQL, cursors, dynamic SQL, ODBC, JDBC, SQLJ, which can be used for accessing databases from other applications. These techniques will be discussed later in the chapter. 5.2 DATA DEFINITION Data definition language (DDL) commands are used for defining relation schemas, deleting relations, creating indexes, and modifying relation schemas. In this section, DDL commands are discussed and an overview of the basic data types in SQL is provided. In addition, it discusses how integrity constraints can be specified for a relation. 5.2.1 Concept of Schema and Catalog in SQL In earlier versions of SQL, there was no concept of relational database schema, thus, all the database objects were considered to be a part of same database schema. It implies that no two database objects can share same name, since they all are part of the same name space. Due to this, all users of the database have to coordinate with each other to ensure that they use different names for database objects. However, the present day database systems provide a three level hierarchy for naming different objects of relational database. This hierarchy comprises catalogs, which in turn consists of schemas, and various database objects are incorporated within the schema. For example, any relation can be uniquely identified as: catalog_name.schema_name.relation_name NOTE Schema is identified by a schema name, and its elements include relations, constraints, views, domains, etc. Database system can have multiple catalogs, and users working with different catalogs can use the same name for database objects without any clashes. Each user of database system is associated with a default catalog and schema, and whenever the user connects with the database (by providing the unique combination of user name and password), he gets connected with the default catalog and schema. In SQL, one can create schema by using the CREATE SCHEMA command, including definitions of all the elements of schema. Alternatively, the schema can be given a name and authorization identifier to indicate the user who is owner of schema, and the elements of schema can be specified later. For example, to create Online Book schema owned by the user identified by authorization identifier Smith, the command can be specified as CREATE SCHEMA OnlineBook AUTHORIZATION Smith; The schema can be deleted by using the DROP SCHEMA command. The privilege to create or drop schemas, relations, and other components of database is granted to relevant user by the system administrator. Integrity constraints like referential integrity can be defined between relations only if they belong to schemas within the same catalog. 107
5.2.2 Data Types Data type identifies the type of data to be stored in an attribute of a relation and also specifies associated operations for handling the data. Data type defined for an attribute associates a set of properties with that attribute. These properties cause attributes of different data types to have different set of operations that can be performed on these attributes. The common data types supported by standard SQL are ? NUMERIC(p, s): used to represent data as floating-point number like 17.312, 27.1204, etc. The number can have p significant digits (including sign) and s number of the p digits can be present on the right of decimal point. For example, data type specified as NUMERIC(5, 2) indicates that value of an attribute can be of form 332.32, where as number of the forms 32.332 or 0.332 are not allowed. ? INT or INTEGER: used to represent data as a number without a decimal point. The size of the data is machine dependent. ? SMALLINT: used to represent data as a number without a decimal point. It is a subset of the INTEGER so the default size is usually smaller than INT. ? CHAR(n) or CHARACTER(n): used to represent data as fixed-length string of characters of size n. In case of fixed- length strings, a shorter string is padded with blank characters to the right. For example, if the value ABC is to be stored for an attribute with data type CHAR(8), the string is padded with five blanks to the right. Padded blanks are ignored during comparison operation. ? VARCHAR(n) or CHARACTER VARYING: used to represent data as variable length string of characters of maximum size n. In case of variable length string, a shorter string is not padded with blank characters. ? DATE and TIME: used to represent data as date or time. The DATE data type has three components, namely,
year, month, and day in the form YYYY-MM-DD. The TIME data type also have three components, namely, hours, minutes, and seconds in the form HH:MM:SS. ?

BOOLEAN: used to represent the third value unknown, in addition to true and false values, because of the presence of null values in SQL. ? TIMESTAMP: used to represent data consisting of both date and time. The TIMESTAMP data type has six components, year, month, day, hour, minute, and second in the form YYYY-MM-DDHH:MM:SS[.sF], where F is the fractional part of the second value. In addition to these data types, there are many more data types supported by SQL like CLOB (CHARACTER LARGE OBJECT), BLOB (BINARY LARGE OBJECT), etc. These data types are rarely used and hence, are not discussed here. In addition to built-in data types, user-defined data types can also be created. The user-defined data type can be created using the CREATE DOMAIN command. For example, to create user-defined data type Vchar, the command can be specified as CREATE DOMAIN Vchar AS VARCHAR(15); Now, Vchar can be used as data type for any attribute for which data type VARCHAR(15) is to be defined. After declaring such data type, it becomes easier to make changes in the domain that is being used by numerous attributes in a relation schema. 5.2.3 CREATE TABLE Command The CREATE TABLE command is used to define a new relation, its attributes and its data types. In addition, various constraints like key, entity integrity, and referential integrity constraints can also be specified. The syntax for CREATE TABLE command is shown here. CREATE TABLE &gt;table_name&lt; (&gt;attribute 1 &lt; &gt;data_type 1 &lt; [constraint 1], &gt;attribute 2 &lt; &gt;data_type 2 &lt; [constraint 2], : &gt;attribute n &lt; &gt;data_type n &lt; [constraint n], [table_constraint 1] 108
: [table_constraint n]); where, table_name is the name of new relation attribute i is the attribute of relation data_type i is the data type of values of the attribute constraint i is any of the column-level constraints defined on the corresponding attribute table_constraint i is any of the table-level constraints For example, the command to create BOOK relation whose schema is BOOK(ISBN, Book_title, Category, Price, Copyright_date, Year, Page_count, P_ID) can be specified as CREATE TABLE BOOK (ISBN VARCHAR(15), Book_title VARCHAR(50), Category VARCHAR(20), Price NUMERIC(6,2), Copyright_ date NUMERIC(4), Year NUMERIC(4), Page_count NUMERIC(4), P_ID VARCHAR(4)); Once the relation is created, its structure can be viewed by using the DESCRIBE (or DESC) command. For example, the command to view the structure of BOOK relation can be specified as DESCRIBE BOOK OR DESC BOOK As a result of this command, the structure of the relation BOOK is displayed as shown here. Name Null? Type ------------------------------------ -------- ------------ ISBN VARCHAR2(15) BOOK_TITLE VARCHAR2(50) 109
CATEGORY VARCHAR2(20) PRICE NUMBER(6,2) COPYRIGHT_DATE NUMBER(4) YEAR NUMBER(4) PAGE_COUNT NUMBER(4) P_ID VARCHAR2(4) In this example, the relation BOOK is created without specifying any constraints. Since no constraints are defined, invalid values can be entered in the relation. To avoid such a situation, it is necessary to define constraints within the definition of the relation. 5.2.4 Specifying Constraints As discussed earlier, constraints are required to maintain the integrity of the data, which ensures that the data in database is consistent, correct, and valid. These constraints can be specified within the definition of a relation. These include key, integrity, and referential constraints along with the restrictions on attribute domains and null values. This section discusses how different types of constraints can be specified at the time of creation of relation. PRIMARY KEY Constraint This constraint ensures that attribute declared as primary key cannot have null value and no two tuples can have same value for primary key attribute. In other words, the values in the primary key attribute are not null and are unique. If a primary key has a single attribute, the PRIMARY KEY can be applied as a column-level as well as table-level constraint. The syntax of PRIMARY KEY constraint when applied as a column-level constraint is given here. &gt;attribute&lt; &gt;data_type&lt; PRIMARY KEY The syntax of PRIMARY KEY constraint when applied as a table-level constraint is PRIMARY KEY (&gt;attribute&lt;) For example, the attribute ISBN of BOOK relation can be declared as a primary key as shown here. ISBN VARCHAR(15) PRIMARY KEY OR PRIMARY KEY (ISBN) If a primary key has more than one attribute, the PRIMARY KEY constraint is specified as table-level constraint. The syntax of PRIMARY KEY constraint when applied on more than one attribute is PRIMARY KEY (&gt;attribute 1 &lt;, &gt;attribute 2 &lt;, ..., &gt;attribute n &lt;) For example, attributes ISBN and R_ID of REVIEW relation can be declared as composite primary key as shown here. PRIMARY KEY (ISBN, R_ID) UNIQUE Constraint The UNIQUE constraint ensures that the set of attributes have unique values, that is, no two tuples can have same value in the specified attributes. Like PRIMARY KEY constraint, UNIQUE constraint can be applied as a column-level as well as table-level constraint. The syntax of UNIQUE constraint when applied as a column-level constraint is given here. 110

&gt;attribute&lt; &gt;data_type&lt; UNIQUE The syntax of UNIQUE constraint when applied as a table-level constraint is UNIQUE (&gt;attribute&lt;) For example, the UNIQUE constraint on attribute Pname of the relation PUBLISHER can be specified as shown here. Pname VARCHAR(50) UNIQUE OR UNIQUE (Pname) When UNIQUE constraint is applied on more than one attribute it is specified as table-level constraint. The syntax of UNIQUE constraint when applied on more than one attribute is UNIQUE (&gt;attribute 1 &lt;, &gt;attribute 2 &lt;;, ..., &gt;attribute n &lt;) For example, the UNIQUE constraint on attributes Pname and Address of relation PUBLISHER can be specified as shown here. UNIQUE (Pname, Address) CHECK Constraint This constraint ascertains that the value inserted in an attribute must satisfy a given expression. In other words, it is used to specify the valid values for a certain attribute. The syntax of CHECK constraint when applied as a column-level constraint is given here. &gt;attribute&lt; &gt;data_type&lt; CHECK (&gt;expression&lt;) Learn More SQL allows setting default value for an attribute by adding the DEFAULT &gt;default_value&lt; clause to the definition of an attribute in CREATE TABLE command. For example, the CHECK constraint for ensuring that the value of attribute Price of the relation BOOK is greater than $20 can be specified as Price NUMERIC(6,2) CHECK (Price&lt;20) The CHECK constraint when specified as table-level constraint can be given a separate name that allows referring to the constraint whenever needed. The syntax of CHECK constraint when applied as a table-level constraint is CONSTRAINT &gt;constraint_name&lt; CHECK (&gt;expression&lt;) For example, the constraint on the attribute Price of the BOOK relation can be given a name as shown here. CONSTRAINT Chk_price CHECK (Price &lt; 20) Constraints can also be applied on more than one attribute simultaneously. For example, the constraint that the Copyright_date must be either less than or equal to the Year (publishing year) can be specified as CONSTRAINT Chk_date CHECK (Copyright_date &gt;= Year) When CHECK constraint is applied to a domain, it allows specifying a condition that must be satisfied by any value assigned to an attribute whose type is the domain. For example, the CHECK constraint can ensure that the domain, say dom, allows values between 20 and 200 as shown here. 111
CREATE DOMAIN dom AS NUMERIC(5,2) CONSTRAINT chk_dom CHECK (VALUE BETWEEN 20 and 200) If the domain dom is assigned to the attribute Price of a BOOK relation, it ensures that the attribute Price must have values between 20 and 200. As a result, if a value that is not between 20 and 200 is inserted into the Price attribute, an error occurs. A domain can also be restricted to contain a specified set of values using IN clause with CHECK constraint. For example, the CHECK constraint ensuring that the domain, say dom1, allows values within a list of certain values can be specified as CREATE DOMAIN dom1 CHAR(20) CONSTRAINT chk_dom1 CHECK (VALUE IN ('Textbook', 'Language Book', 'Novel')) NOT NULL Constraint The NOT NULL constraint is used to specify that an attribute will not accept null values. For example, consider a tuple in the BOOK relation where the attribute Page_count contains a null value. Such a tuple provides incomplete information of that particular book. In such a case, using the NOT NULL constraint does not permit null value. The syntax of NOT NULL constraint is given here. &gt;attribute&lt; &gt;data_type&lt; NOT NULL For example, the NOT NULL constraint for the attribute Page_count of BOOK relation can be specified as shown here. Page_count Numeric(4) NOT NULL The NOT NULL constraint can be specified using CHECK constraint. For example, the NOT NULL constraint on the attribute Book_title can be specified using the CHECK constraint as shown here. Book_title VARCHAR(50) CHECK (Book_title IS NOT NULL) The NOT NULL constraint ensures that the attribute of a particular domain is not permitted to take null values. For example, a domain, say dom2, can be restricted to take non-null values as shown here. CREATE DOMAIN dom2 VARCHAR(20) NOT NULL Keypoint: The NOT NULL constraint can be specified only as column-level constraint and not as table-level constraint. FOREIGN KEY Constraint This constraint ensures that the foreign key value in the referencing relation must exist in the primary key attribute of the referenced relation, that is, foreign key references the primary key attribute of referenced relation. A FOREIGN KEY constraint can be applied as a column-level as well as table-level constraint. The syntax of FOREIGN KEY constraint when applied as a column-level constraint is given here. &gt;attribute&lt; &gt;data_type&lt; REFERENCES &gt;referenced_relation&lt; (&gt;key_attribute&lt;) where, key_attribute is the primary key of the referenced relation For example, the attribute P_ID of relation BOOK can be specified as foreign key, which refers to the primary key P_ID of relation PUBLISHER as shown here. P_ID VARCHAR(4) REFERENCES PUBLISHER(P_ID) When FOREIGN KEY constraint is applied on more than one attribute it is specified as table-level constraint. The syntax of FOREIGN KEY constraint defined on more than one attribute is given here. 112
FOREIGN KEY (&gt;attribute 1 &lt;) REFERENCES &gt;referenced_relation&lt; (&gt;key_attribute_1&lt;) : FOREIGN KEY (&gt;attribute n &lt;) REFERENCES &gt;referenced_relation&lt; (&gt;key_attribute_n&lt;) For example, attributes ISBN and R_ID of relation REVIEW can be declared as foreign keys as shown here. FOREIGN KEY (ISBN) REFERENCES BOOK(ISBN) FOREIGN KEY (R_ID) REFERENCES AUTHOR(A_ID) All the constraints discussed here can be defined together for a relation. The CREATE TABLE command for the relations of Online Book database along with the required constraints is specified here. CREATE TABLE PUBLISHER (P_ID VARCHAR(4), Pname VARCHAR(50) NOT NULL, Address VARCHAR(50), State VARCHAR(15), Phone VARCHAR(20), Email_id VARCHAR(30), PRIMARY KEY(P_ID)); CREATE TABLE BOOK (ISBN VARCHAR(15), Book_title VARCHAR(50) NOT NULL, Category VARCHAR(20), Price NUMERIC(6,2), Copyright_date NUMERIC(4), Year NUMERIC(4), Page_count NUMERIC(4), P_ID VARCHAR(4) NOT NULL, CONSTRAINT Chk_price CHECK (Price BETWEEN 20 AND 200), PRIMARY KEY(ISBN), FOREIGN KEY(P_ID) REFERENCES PUBLISHER(P_ID) 113

); CREATE TABLE AUTHOR (A_ID VARCHAR(4), Aname VARCHAR(30) NOT NULL, State VARCHAR(15), City VARCHAR(15), Zip VARCHAR(10), Phone VARCHAR(20), URL VARCHAR(30), PRIMARY KEY (A_ID)); CREATE TABLE AUTHOR_BOOK (A_ID VARCHAR(4) NOT NULL, ISBN VARCHAR(15) NOT NULL, FOREIGN KEY(A_ID) REFERENCES AUTHOR(A_ID), FOREIGN KEY(ISBN) REFERENCES BOOK(ISBN)); CREATE TABLE REVIEW (R_ID VARCHAR(4) NOT NULL, ISBN VARCHAR(15) NOT NULL, Rating NUMERIC(2), PRIMARY KEY(R_ID, ISBN), CONSTRAINT Chk_rating CHECK (Rating BETWEEN 1 AND 10), FOREIGN KEY(R_ID) REFERENCES AUTHOR(A_ID), FOREIGN KEY(ISBN) REFERENCES BOOK(ISBN)); 5.2.5 ALTER TABLE Command Once the relation is created, it might be required to make changes in the structure of a relation as per the needs of a user. The changes can be in the form of adding a new attribute, redefining attribute, or dropping attribute from a relation. To 114
meet such requirements, ALTER TABLE command is used. In general, ALTER TABLE command is used for the following requirements. ? to add an attribute ? to modify the definition of an attribute ? to drop an attribute ? to add a constraint ? to drop a constraint ? to rename attribute and relation Adding an Attribute The new attribute can be added to the relation by using the ADD clause of ALTER TABLE command. The syntax to add new attribute in an existing relation is given here. ALTER TABLE &gt;table_name&lt; ADD &gt;attribute&lt; &gt;data_type&lt;; For example, the command to add a new attribute Pname in the relation BOOK can be specified as ALTER TABLE BOOK ADD Pname VARCHAR(10); If no default clause is specified, this attribute will have null values in all the tuples, hence NOT NULL constraint is not allowed in this case. Modifying an Attribute Any attribute of a relation can be modified by using the MODIFY clause of ALTER TABLE command. It can be used to change either its data type or size or both. The attribute to be modified must be empty before modification. The syntax to modify an attribute of a relation is given here. ALTER TABLE &gt;table_name&lt; MODIFY &gt;attribute&lt; &gt;data_type&lt;; For example, the command to change the data type and size of the attribute Pname can be specified as ALTER TABLE BOOK MODIFY Pname VARCHAR(50); Dropping an Attribute Sometimes, it is required to remove attribute, which is no longer required from a relation. The DROP COLUMN clause of ALTER TABLE command can be used to remove undesirable attributes from a relation. The syntax to remove an attribute from an existing relation is given here. ALTER TABLE &gt;table_name&lt; DROP COLUMN &gt;attribute&lt;; For example, the command to remove the attribute Pname from the relation BOOK can be specified as ALTER TABLE BOOK DROP COLUMN Pname; Whenever a particular attribute is dropped, the data stored in that attribute and its associated constraints are dropped. Adding a Constraint Different types of constraints can be added to the definition of an already existing relation. The syntax to add a constraint using ADD clause is given here. ALTER TABLE &gt;table_name&lt; ADD [CONSTRAINT &gt;constraint_name&lt;] &gt;Cons&lt;; 115
where, CONSTRAINT is a keyword constraint_name is a name given to constraint Cons can be any of the constraints discussed earlier Consider the following examples, in which different types of constraints are added for the different attributes of BOOK relation. ALTER TABLE BOOK ADD CHECK (Book_title &gt;&lt; ''); ALTER TABLE BOOK ADD CONSTRAINT Cons_1 UNIQUE (ISBN); ALTER TABLE BOOK ADD FOREIGN KEY (P_ID) REFERENCES PUBLISHER; The NOT NULL constraint is added differently as it cannot be specified as table constraint. For example, the command to apply this constraint on the attribute Page_count can be specified as ALTER TABLE BOOK ALTER COLUMN Page_count SET NOT NULL ; It is also possible to modify the definition of attribute by defining a new default value for an attribute, as shown here. ALTER TABLE BOOK ALTER COLUMN Category SET DEFAULT 'Novel'; Dropping a Constraint Any of the constraint defined can be dropped, provided it is given a name when specified. For example, the constraint Cons_1 specified on the attribute ISBN, can be dropped as shown here. ALTER TABLE BOOK DROP CONSTRAINT Cons_1; In addition, the DEFAULT and NOT NULL constraint can be dropped, as shown here. ALTER TABLE BOOK ALTER COLUMN Category DROP DEFAULT ; ALTER TABLE BOOK ALTER COLUMN Page_count DROP NOT NULL ; Renaming an Attribute The name of an attribute of a relation can be modified by using the RENAME COLUMN ... TO clause. The syntax to modify the name of an attribute is given here.
ALTER TABLE &gt;table_name&lt; RENAME COLUMN &gt;old_name&lt; TO &gt;new_name&lt;; For example,
the command to modify the name of an attribute Page_count can be specified as ALTER TABLE BOOK RENAME COLUMN Page_count TO P_count; Renaming a Relation In addition to renaming an attribute, the name of a relation can also be modified using the RENAME TO clause. The syntax to rename a relation is given here. ALTER TABLE &gt;old_table_name&lt; RENAME TO &gt;new_table_name&lt;; For example, the command to rename the relation BOOK to new name can be specified as ALTER TABLE BOOK RENAME TO Book_detail; 5.2.6 DROP TABLE Command 116

The DROP TABLE command is used to remove an already existing relation, which is no more required as a part of a database. The syntax to remove a relation is given here. DROP TABLE &gt;table_name&lt;; For example, the command to remove the relation Book_detail can be specified as DROP TABLE Book_detail; As a result of this command, all the tuples stored in Book_detail as well as relation itself is permanently removed and cannot be recovered. Hence, the relation Book_detail cannot be referred to for any purpose. The two clauses that can be used with DROP TABLE command are CASCADE and RESTRICT. If CASCADE clause is used, all constraints and views that reference the relation to be removed are also dropped automatically. On the other hand, the RESTRICT clause, prevents a relation to be dropped if it is referenced by any of the constraints or views. These clauses can be used with DROP TABLE command as shown here. DROP TABLE Book_detail CASCADE ; DROP TABLE Book_detail RESTRICT ; 5.3 DATA MANIPULATION LANGUAGE Data manipulation language (DML) commands are used for retrieving and manipulating data stored in the database. Basically it comprises of different actions, which are given here. ? retrieval of data
stored in the database ? insertion of data into the database ? modification of data stored in the database ? deletion of data stored in the database
The basic retrieval and manipulating commands are SELECT, INSERT, UPDATE, and DELETE. For these commands, the relation must already exist. 5.3.1 SELECT Command The SELECT command is the only data retrieval command in SQL. It is used to retrieve the subset of tuples or attributes from one or more relations. There are different ways and combinations in which SELECT command can be used. The syntax of SQL command is given here. SELECT &gt;attribute 1 &lt;, &gt;attribute 2 &lt;;, ..., &gt;attribute n &lt; FROM &gt;table_name&lt;; Here, attribute i is an attribute of relation table_name. For example, the command to retrieve the attributes ISBN, Book_title, and Category from relation BOOK can be specified as SELECT ISBN, Book_title, Category FROM BOOK; The order of the attributes appearing in the command can be different from the order of attributes in the relation. For example, the command to retrieve the attributes ISBN, Category, Book_title, Page_count, and Price can be specified as SELECT ISBN, Category, Book_title, Page_count, Price FROM BOOK; In this example, the order of attributes is different from the order in which they are appearing in the relation. When all the attributes of a relation has to be retrieved, then instead of specifying list of all attributes in the SELECT command, the symbol asterisk (*) denoting "all attributes" can be used as shown here. 117 SELECT * FROM BOOK; The result of the SQL command may consist of duplicate tuples, as SQL does not automatically eliminate duplicate tuples from the resultant relations. For example, the command to retrieve Category of books from the relation BOOK can be specified as SELECT Category FROM BOOK; The result of this command consists of tuples having duplicate values for the attribute Category. To eliminate duplicate tuples from the resultant relation, the keyword DISTINCT is used in SELECT command. Hence, the command to display only the unique values for the attribute Category can be specified as SELECT DISTINCT Category FROM BOOK; Instead of DISTINCT keyword, if the keyword ALL is specified, the result retains the duplicate tuples. The result is same as when neither DISTINCT nor ALL keywords are specified. The ALL keyword can be used as shown here. SELECT ALL Category FROM BOOK; The SELECT command can be used to perform simple numeric computations on the data stored in a relation. SQL allows using scalar expressions and constants along with the attribute list. For example, the command to display the incremented value of price of books by 10% along with the attributes Book_title, Category, and Price can be specified as SELECT Book_title, Category, Price, Price+Price*.10 FROM BOOK; WHERE clause The real life database might consist of a large number of tuples and it is not required to view all the tuples every time. Moreover, most of the time only subset of tuples is required to be viewed or processed. The criteria to determine the tuples to be retrieved is specified by using the WHERE clause. The syntax of SELECT command with WHERE clause is given here. SELECT &gt;attribute 1 &lt;, &gt;attribute 2 &lt;;, ..., &gt;attribute n &lt; FROM &gt;table_name&lt; WHERE &gt;condition&lt;; When a WHERE clause is present, the entire relation is scanned, one tuple at a time to determine whether it satisfies the condition or not. A particular tuple is retrieved only if it satisfies the condition specified in WHERE clause. For example, the command to retrieve the attributes Book_title, Category, and Price of those books from BOOK relation whose Category is Novel can be specified as SELECT Book_title, Category, Price FROM BOOK WHERE Category = 'Novel'; The relational operators (=, &gt;&lt;, &gt;, &gt;=, &lt;, &lt;=) can be used to specify the conditions in the WHERE clause for selecting tuples from a relation. Moreover, more than one condition can be concatenated to give a more specific condition by using any of the logical operators AND, OR, and NOT. For example, the command to retrieve Book_title and Price of those books whose Category is Language Book and Page_count is greater than 400 from BOOK relation can be specified as 118

SELECT Book_title, Price FROM BOOK WHERE Category = 'Language Book' AND Page_count&lt;400; Consider the queries covered in Chapter 04. These queries can be expressed here using the SELECT command along with the WHERE clause as shown here. Query 1: Retrieve city, phone, and url of the author whose name is Lewis Ian. SELECT City, Phone, URL FROM AUTHOR WHERE Aname = 'Lewis Ian'; Query 2: Retrieve name, address, and phone of all the publishers located in New York state. SELECT Pname, Address, Phone FROM PUBLISHER WHERE State = 'New York'; Query 3: Retrieve title and price of all the textbooks with page count greater than 600. SELECT Book_title, Price FROM BOOK WHERE Category = 'Textbook' AND Page_count&lt;600; Query 4: Retrieve ISBN, title, and price of the books belonging to either novel or language book category. SELECT ISBN, Book_title, Price FROM BOOK WHERE Category = 'Novel' OR Category = 'Language Book'; BETWEEN Operator The BETWEEN comparison operator can be used to simplify the condition in WHERE clause that specifies numeric values based on ranges. For example, the command to retrieve details of all the books with price between 20 and 30 can be specified as SELECT * FROM BOOK WHERE Price BETWEEN 20 AND 30; Similarly, the NOT BETWEEN operator can also be used to retrieve tuples that do not belong to the specified range. IN operator The IN operator is used to specify a list of values. The IN operator selects values that match any value in a given list of values. For example, the command to retrieve the book details belonging to the categories Textbook or Novel can be specified as SELECT * FROM BOOK WHERE Category IN ('Textbook', 'Novel'); Similarly, NOT IN operator can also be used to retrieve tuples that do not belong to the specified list. 119
Aliasing and Tuple Variables The attributes of a relation can be given alternate name using AS clause in SELECT command. Note that the alternate name is provided within the query only. For example, consider the command given here. SELECT Book_title AS Title, P_ID AS Publisher_ID FROM BOOK; In this command, attribute Book_title is renamed as Title and P_ID as Publisher_ID. An AS clause can also be used to define tuple variables. A tuple variable is associated with a particular relation and is defined in the FROM clause. For example, to retrieve details of publishers publishing books of language book category, the command can be specified as SELECT P.P_ID, Pname, Address, Phone FROM BOOK AS B, PUBLISHER AS P WHERE P.P_ID = B.P_ID AND Category = 'Language Book'; In this command, tuple variables B and P are defined which are associated with the relations BOOK and PUBLISHER, respectively. The attribute publisher ID is present in both the relations with same name P_ID. Thus, to prevent ambiguity, the attribute name is qualified with the corresponding tuple variable separated by the dot (.) operator. Tuple variable is especially useful for comparing two tuples from the same relation. For example, the command to retrieve title and price of books belonging to the same category as the book titled C++ can be specified as SELECT B1.Book_title, B1.Price FROM BOOK AS B1, BOOK AS B2 WHERE B1.Category = B2.Category AND B2.Book_title = 'C++'; NOTE The relation name itself is the implicitly defined tuple variable, if no other tuple variable is defined for that relation. String Operations In SQL, strings are specified by enclosing them in single quotes. Pattern matching is the most commonly used operation on strings. SQL provides LIKE operator, which allows to specify comparison conditions on only parts of the strings. Different patterns are specified by using two special wildcard character, namely, percent (%) and underscore (_). The % character is used to match substring with any number of characters, whereas, _ is used to match substring with only one character. Pattern matching is case sensitive as uppercase characters are considered different from lowercase characters. That is, the character s is different from the character S. To understand, how pattern matching is handled, consider these examples. ? 'A%' matches any string beginning with character A ? '%A' matches any string ending with character A ? 'A%A' matches any string beginning and ending with character A ? '%AO%' matches any string with character AO appearing anywhere in a string as substring ? 'A_ _' matches any string beginning with character A and followed by exactly two characters ? '_ _A' matches any string ending with character A and preceded by exactly two characters ? 'A_ _ _D' matches any string beginning with character A, ending with the character D and with exactly three characters in between Consider the following queries to illustrate the use of LIKE operator. The command to retrieve details of all the authors, whose name begins with the characters Jo can be specified as SELECT * FROM AUTHOR WHERE Aname LIKE 'Jo%'; 120

The command to retrieve details of all the authors, whose name ends with the characters in can be specified as SELECT * FROM AUTHOR WHERE Aname LIKE '%in'; The command to retrieve details of all the authors, whose name begins with the characters Jo and ends with the characters in can be specified as SELECT * FROM AUTHOR WHERE Aname LIKE 'Jo%in'; The command to retrieve details of all the authors, whose name begins with the characters James followed by exactly five characters, can be specified as SELECT * FROM AUTHOR WHERE Aname LIKE 'James_ _ _ _ _'; The special pattern characters (% and _) can be included as a literal in the string by preceding the character by an escape character. The escape character is specified after the string using the ESCAPE keyword. Any character not appearing in a string to be matched can be defined as an escape character. For example, consider the following patterns, in which backslash (\) is used as an escape character. ? LIKE 'A\%b%' ESCAPE '\', searches the strings beginning with the characters A%b ? LIKE 'A_b%' ESCAPE '\', searches the strings beginning with the characters A_b ? LIKE 'A\\b%' ESCAPE '\', searches the strings beginning with the characters A\b Things to Remember SQL allows to perform various functions on character strings like concatenation of strings (using '||' operator), extracting substrings, computing length of a string, converting case of strings uppercase (using UPPER() and LOWER() function), etc. In order to search the strings not matching the patterns defined, NOT LIKE operator can be used. For example, the command to retrieve details of all the authors, whose name does not begins with the characters Jo can be specified as SELECT * FROM AUTHOR WHERE Aname NOT LIKE 'Jo%'; Set Operations As discussed in Chapter 04, there are various set operations that can be performed on relations, like, union, intersection, and difference. In SQL, these operations are implemented using UNION, INTERSECT, and MINUS operations, respectively. The UNION operation is used to retrieve tuples from more than one relation and it also eliminates duplicate records. For example, the command to find the union of all the tuples with Price less than 40 and all the tuples with Price greater than 30 from the BOOK relation can be specified as (SELECT * FROM BOOK WHERE Price&gt;40) UNION (SELECT * FROM BOOK 121
WHERE Price&lt;30); The INTERSECT operation is used to retrieve common tuples from more than one relation. For example, the command to find the intersection of all the tuples with Price less than 40 and all the tuples with Price greater than 30 from the BOOK relation can be specified as (SELECT * FROM BOOK WHERE Price&gt;40) INTERSECT (SELECT * FROM BOOK WHERE Price&lt;30); The MINUS operation is used to retrieve those tuples present in one relation, which are not present in other relation. For example, the command to find the difference (using MINUS operation) of all the tuples with Price less than 40 and all the tuples with Price greater than 30 from the BOOK relation can be specified as (SELECT * FROM BOOK WHERE Price&gt;40) MINUS (SELECT * FROM BOOK WHERE Price&lt;30); The UNION, INTERSECT, and MINUS operations automatically eliminate the duplicate tuples from the result. However, if all the duplicate tuples are to be retained, the ALL keyword can be used with these operations. For example, the command to retain duplicate tuples during UNION operation can be specified as (SELECT * FROM BOOK WHERE Price&gt;40) UNION ALL (SELECT * FROM BOOK WHERE Price&lt;30); Similarly, ALL keyword can be used along with the INTERSECT and MINUS operations. NOTE During all the set operations, the resultant relations of both the participating SELECT commands must be union compatible. That is, both the resultant relations must have same number of attributes having similar data types and order. Ordering the Tuples Sometimes it may be required to display tuples arranged in a sorted order. SQL provides the ORDER BY clause to arrange the tuples of relation in some particular order. The tuples can be arranged on the basis of the values of one or more attributes. For example, the command to display the tuples of the relation BOOK, on the basis of Price attribute can be specified as SELECT * FROM BOOK ORDER BY Price; 122
By default, the ORDER BY clause arranges the tuples in ascending order on the basis of values of specified attribute. In addition, the ASC keyword can also be used to explicitly mention the order to be ascending. However, the tuples can be sorted in descending order by using the DESC keyword. For example, the command to sort tuples of the relation BOOK on the basis of Price attribute in the descending order can be specified as SELECT * FROM BOOK ORDER BY Price DESC ; Tuples can also be sorted on the basis of more than one attribute. This can be done by specifying more than one attribute in ORDER BY clause. In ORDER BY clause, more than one attributes can also be specified on the basis of which tuples are to be sorted. For example, to arrange the sort of the relation BOOK, on the basis of two attributes Category and Price, the command can be specified as SELECT * FROM BOOK ORDER BY Category, Price; This command first arranges the tuples in ascending order on the basis of the Category attribute and then the tuples within a particular category are arranged in the ascending order of the Price attribute. The order of two attributes need not be the same. They can be different. For example, to arrange the tuples of relation BOOK in the descending order of Category and then in the ascending order of Price, the command can be specified as SELECT * FROM BOOK ORDER BY Category DESC, Price ASC ; 5.3.2 INSERT Command As discussed earlier, the CREATE TABLE command only defines the structure of a relation and tuples can be inserted in the relation using the INSERT command. This command adds a single tuple at a time in a relation. The syntax to add a tuple in a relation is given here. INSERT INTO &gt;table_name&lt; [(&gt;attribute_list&lt;)] VALUES (&gt;value_list&lt;); The value_list is the list of values to be inserted in the corresponding attributes listed in attribute_list.
The values should be specified in the same order in which the attributes

are listed in attribute_list. In addition, the data type of the corresponding values and the attributes must be same. For example, the command to add a tuple in the relation BOOK can be specified as INSERT INTO BOOK (ISBN, Book_title, Category, Price, Copyright_date, Year, Page_count, P_ID) VALUES ('001-987-760-9', 'C++', 'Textbook', 40, 2004, 2005, 800, 'P001'); In this example, the order and the data type of attributes and corresponding values are same. Alternatively, tuple can be inserted into a relation by omitting the attribute list from the command. That is, the tuple can also be inserted as shown here. INSERT INTO BOOK VALUES ('001-987-760-9', 'C++', 'Textbook', 40, 2004, 2005, 800, 'P001'); The INSERT command also allows user to insert data only for the selected attributes. That is, in INSERT command, values for some of the attributes can be skipped. The attributes skipped in the command are provided with the default values defined for them, otherwise null values are inserted. In this case, the attribute list must be provided. For example, the command to add values for the selected attributes of the relation PUBLISHER can be specified as 123
INSERT INTO PUBLISHER (P_ID, Pname, Address, Phone) VALUES ('P002', 'Sunshine Publishers Ltd.', '45, Second street, Newark', '6548909'); In this example, the values for the attributes State and Email_id are skipped. Hence, these attributes are provided either with the default values or null values. 5.3.3 UPDATE Command Sometimes, there may be a situation to make changes in the values of attributes of the relations. SQL provides UPDATE command for this type of requirement. The
SET clause in the UPDATE command specifies the attributes to be modified and
the new values to be assigned to them. More than one attribute can be specified in the SET clause separated by comma. The WHERE clause is also required to specify the tuples for which the attributes are to be modified, otherwise value for all the tuples in the relation will be modified. While modifying the values of attributes all the constraints must be satisfied, otherwise the update is not done. For example, the command to modify the City to New York for the author whose name is Lewis Ian can be specified as UPDATE AUTHOR SET City = 'New York' WHERE Aname = 'Lewis Ian'; The command to modify State and Phone for the publisher Bright Publications can be specified as UPDATE PUBLISHER SET State = 'Georgia', Phone = '27643676' WHERE Pname = 'Bright Publications'; The command to increment the price of all the books by 10 per cent can be specified as UPDATE BOOK SET Price = Price + 0.10 * Price; 5.3.4 DELETE Command The tuples, which are no more required as a part of a relation, can be removed from the relation using the DELETE command. Tuples can be deleted from only one relation at a time. The condition in the WHERE clause of DELETE commands is used to specify the tuples to be deleted. If WHERE clause is omitted, all the tuples of a relation are deleted; however, the relation remains in the database as an empty relation. For example, to delete tuples of the BOOK relation, whose Page_Count is less than 50, the command can be specified as DELETE FROM BOOK WHERE Page_count &gt;50; Consider another example, to delete those tuples from the BOOK relation, whose category is Novel, the command can be specified as DELETE FROM BOOK WHERE Category = 'Novel'; 5.4 COMPLEX QUERIES IN SQL The queries discussed so far are some of the simple queries in SQL. In addition to these, complex queries can be specified using additional features of SQL. Some of these additional features are discussed in this section. 5.4.1 Null Values 124
SQL allows attribute to have null values. The null value usually represents missing value having one of three interpretations––value is unknown, value is not available, or attribute is not applicable for particular tuple. However, SQL does not distinguish between the different meanings of null. The null value in an attribute for a relation can be searched using the IS NULL predicate in the WHERE clause. In all cases, null value for an attribute is checked with same syntax. For example, the command to retrieve the details of publishers not having email ID can be specified as SELECT * FROM PUBLISHER WHERE Email_id IS NULL ; The predicate IS NOT NULL can be used to check whether an attribute contains non-null value. For example, to retrieve the details of publishers having email ID (Email_id is not null), the command can be specified as SELECT * FROM PUBLISHER WHERE Email_id IS NOT NULL ; In addition, the conditions in WHERE clause may involve logical operators AND, OR, and NOT. Hence, when null value is involved, the definition of the logical operators is extended with new value unknown. In case of NOT operator, the condition, say A, whose value is unknown, results in unknown value. Whereas, the AND and OR operator involves two (or more) conditions. For this, consider a WHERE clause involving two conditions A and B, and the result of any one of them, say B, is unknown, then the condition evaluates as follows: ? Conditions involving AND operator:
o If
the value of A is true or unknown, the result is unknown. o
If

the value of A is false, the result is false. ? Conditions involving OR operator: o If the value of A is false or unknown, the result is unknown. o If the value of A is true, the result is true. 5.4.2 Aggregate Functions Aggregate functions process set of values taken as input and return a single value as a result. SQL provides five built-in aggregate functions, namely, AVG, MIN, MAX, SUM and COUNT. The SUM and AVG functions works for numeric values only, whereas other functions can work for numeric as well as non- numeric values, like strings, date, time, etc. For example, the command to find the average price of books in BOOK relation can be specified as SELECT AVG (Price) FROM BOOK; This command calculates the average value of price of all the books. To calculate the average price of only textbooks, the command can be specified as SELECT AVG (Price) FROM BOOK WHERE Category='Textbook'; Consider another example to find the maximum price of the book belonging to Novel category, the command can be specified as SELECT MAX (Price) FROM BOOK WHERE Category='Novel'; 125
The COUNT(*) function is used to count the total number of tuples in the resultant relation, whereas, COUNT() is used to count the number of non-null values in a particular attribute. For example, the command to find the total number of tuples in the relation PUBLISHER can be specified as SELECT COUNT (*) FROM PUBLISHER; To find the number of non-null values in the attribute Email_id, the command can be specified as SELECT COUNT (Email_id) FROM PUBLISHER; Consider another example, to find the number of non-null values in the attribute Category of BOOK relation, the command can be specified as SELECT COUNT (Category) FROM BOOK; This command counts the total number of non-null values in the attribute Category, including duplicate values also. However, if duplicate values are to be eliminated, the DISTINCT keyword can be used and the command can be specified as SELECT COUNT (DISTINCT Category) FROM BOOK; 5.4.3 GROUP BY and HAVING Clause In many situations, it is required to apply the aggregate function on a group of tuples from a relation rather on the whole relation. The tuples in the relation can be divided on the basis of the values of one or more attribute. The tuples belonging to a particular group have same value for the attribute on the basis of which grouping is done. The GROUP BY clause can be used in SELECT command to divide the relation into groups on the basis of values of one or more attributes. After dividing the relation into groups, the aggregate functions can be applied on the individual group independently. The aggregate functions are performed separately for each group and return the corresponding result value separately. For example, the command to calculate average price for each category of book in the BOOK relation can be specified as Learn More Using GROUP BY clause, one can create groups within groups known as nested grouping. An Up to 10 level of nesting is allowed in a GROUP BY expression. SELECT AVG (Price) FROM BOOK GROUP BY Category; To calculate maximum, minimum, and average price of the books published by each publisher, the command can be specified as SELECT MAX (Price), MIN (Price), AVG (Price) FROM BOOK GROUP BY P_ID; Conditions can be placed on the groups using HAVING clause. Note that the HAVING clause places conditions on groups, whereas, WHERE clause is used to place conditions on the individual tuples. Another difference between WHERE and HAVING clause is that, conditions specified in the WHERE clause cannot include aggregate functions, whereas, HAVING clause can. For example, the command to retrieve the book categories for which number of books published is less than 5 can be specified as 126
SELECT Category, COUNT (*) FROM BOOK GROUP BY Category HAVING COUNT (*) &gt;5; More than one condition can be specified in the HAVING clause by using logical operators. For example, the command to retrieve average price and average page count for each category of books with average price greater than 30 and average page count less than 900 can be specified as SELECT Category, AVG (Price), AVG (Page_count) FROM BOOK GROUP BY Category HAVING AVG (Price)&lt;30 AND AVG (Page_count)&gt;900; Consider another example, the command to retrieve average price for each category of book with minimum price greater than 30 can be specified as SELECT Category, AVG (Price) FROM BOOK GROUP BY Category HAVING MIN (Price)&lt;30; The IN and the BETWEEN operators can also be used in the HAVING clause. To understand the use of these operators, consider the commands given here. SELECT Category, AVG (Price), MIN (Page_count) FROM BOOK GROUP BY Category HAVING Category IN ('Textbook', 'Novel'); SELECT Category, MIN (Price), MAX (Price) FROM BOOK GROUP BY Category HAVING AVG (Page_count) BETWEEN 300 AND 1000; While using GROUP BY and HAVING clause, some errors may occur. These errors result from the illegal use of group queries. Some of the possible illegal use of group queries is discussed here. ? Combination of non-group and group expressions without using GROUP BY clause. For example, consider the command given here. SELECT Book_title, AVG (Price) FROM BOOK; This command includes non-group attribute Book_title and a group function AVG(Price), which is illegal and results in an error. ? Combination of non-group and group expressions with GROUP BY clause. For example, consider the command given here. SELECT Book_title, Category, AVG (Price) FROM BOOK GROUP BY Category; 127

This command includes non-group attribute Book_title along with group attribute Category and group expression AVG(Price). This command generates an error, as Book_title cannot have same value for all books in a particular Category. This query can be made error free by omitting the attribute Book_title. ? Using group function with WHERE clause instead of HAVING clause. For example, consider the command given here. SELECT Category, AVG (Price) FROM BOOK WHERE MIN (Price)&lt;30 GROUP BY Category; The WHERE clause operates on individual tuples, whereas, group function operates on multiple tuples or on the group of tuples. Hence, group functions cannot be used with WHERE clause. 5.4.4 Joins A query that combines the tuples from two or more relations is known as join query. In such type of queries, more than one relation is listed in the FROM clause. The process of combining data from multiple relations is called joining. For example, consider the command given here. SELECT * FROM BOOK, PUBLISHER; This command returns the cartesian product of the relations BOOK and PUBLISHER, that is, the resultant relation consists of all possible combinations of all tuples from both the relations. It returns relation with cardinality c 1 *c 2 , where c 1 is cardinality of first relation and c 2 is cardinality of second relation. Only selected tuples can be included in the resultant relation by specifying condition on the basis of common attribute of both the relations. The condition on the basis of which relations are joined is known as join condition. For example, the command to retrieve details of both book and publishers, where P_ID attribute in both the relations have identical values can be specified as SELECT * FROM BOOK, PUBLISHER WHERE BOOK.P_ID=PUBLISHER.P_ID; Using alias names for the relations can make this command simple, as shown here. SELECT * FROM BOOK AS B, PUBLISHER AS P WHERE B.P_ID=P.P_ID; This type of join query in which tuples are concatenated on the basis of equality condition is known as equi-join query. The resultant relation of this query consists of two columns for the attribute P_ID having identical values, one from BOOK relation and other from PUBLISHER relation. This can be avoided by explicitly specifying the name of attributes to be included in the resultant relation. Such type of command can be specified as SELECT ISBN, Book_title, Price, B.P_ID, Pname FROM BOOK AS B, PUBLISHER AS P WHERE B.P_ID=P.P_ID; As a result of this command only one column for the attribute P_ID from relation BOOK is displayed in the result. This type of query is known as natural join query. In addition, some additional conditions can also be given to make the selection of tuples more specific apart from the join condition. For example, consider Query 5 (Chapter 04) in which the command to retrieve P_ID, Pname, Address, and Phone of publishers publishing novels can be specified as 128
SELECT P.P_ID, Pname, Address, Phone FROM BOOK AS B, PUBLISHER AS P WHERE B.P_ID=P.P_ID AND Category = 'Novel'; Sometimes, it is required to extract information from more than two relations. In such a case, more than two relations are required to be joined through join query. This can be achieved by specifying names of all the required relations in FROM clause and specifying join conditions in WHERE clause using AND logical operator. For example, consider Query 8 (Chapter 04) in which the command to retrieve title, category and price of all the books written by author Charles Smith can be specified as SELECT Book_title, Category, Price FROM BOOK AS B, AUTHOR AS A, AUTHOR_BOOK AS AB WHERE B.ISBN=AB.ISBN AND AB.A_ID=A.A_ID AND Aname = 'Charles Smith'; Consider some other examples of join queries given here. Query 6: Retrieve title and price of all the books published by Hills Publications. SELECT Book_title, Price FROM BOOK AS B, PUBLISHER AS P WHERE B.P_ID=P.P_ID AND Pname='Hills Publications'; Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks. SELECT Book_title, R_ID, Rating FROM BOOK S B, REVIEW S R WHERE B.ISBN=R.ISBN AND Category = 'Textbook'; Query 9: Retrieve ID, name, url of author, and category of the book C++. SELECT AB.A_ID, Aname, URL, Category FROM BOOK S B, AUTHOR S A, AUTHOR_BOOK S AB WHERE A.A_ID=AB.A_ID AND AB.ISBN=B.ISBN AND Book_title= 'C++'; Query 10: Retrieve book title, price, author name, and url for the publishers Bright Publications. SELECT Book_title, Price, Aname, URL FROM BOOK S B, AUTHOR_BOOK S AB, AUTHOR S A, PUBLISHER S P WHERE B.ISBN=AB.ISBN AND AB.A_ID=A.A_ID ND B.P_ID=P.P_ID AND name='Bright Publications'; 5.4.5 Nested Queries The query defined in the WHERE clause of another query is known as nested query or subquery. The query in which another query is nested is known as enclosing query. The result returned by the subquery is used by the enclosing query for specifying the conditions. In general, several levels of nested queries can be defined. That is, query can be defined inside another query number of times. Things to Remember The two queries are said to be correlated when
condition specified in the WHERE clause of the inner query refers to an attribute of a relation specified in enclosing query. For example, the command to retrieve ISBN, Book_title, and Category of book with minimum price can be specified as 129

SELECT ISBN, Book_title, Category FROM BOOK WHERE PRICE = (SELECT MIN (Price) FROM BOOK); In this command, first the nested query returns a minimum Price from the BOOK relation, which is used by the enclosing query to retrieve the required tuples. Consider another example to retrieve ISBN, Book_title, and Category of book published by the publisher residing in the New York state. SELECT ISBN, Book_title, Category FROM BOOK WHERE P_ID = (SELECT P_ID FROM PUBLISHER WHERE State = 'New York'); SQL provides four useful operators that are generally used with subqueries. These are ANY, ALL, and EXISTS. ? ANY operator compares a value with any of the values in a list or returned by the subquery. This operator returns false value if the subquery returns no tuple. For example, the command to retrieve details of books with price equal to any of the books belonging to Novel category can be specified as ? SELECT ISBN, Book_title, Price ? FROM BOOK ? WHERE Price = ANY (SELECT Price ? FROM BOOK WHERE Category = 'Novel'); In addition to ANY operator, the IN operator can also be used to compare a single value to the set of multiple values. For example, the command to retrieve the details of books belonging to category with page count greater than 300 can be specified as SELECT * FROM BOOK WHERE Category IN (SELECT Category FROM BOOK WHERE Page_count &lt;300); In case the nested query returns a single attribute and a single tuple, the query result will be a single value. In such a situation, the = can be used instead of IN operator for the comparison. ? ALL operator compares a value to every value in a list returned by the subquery. For example, the command to retrieve details of books with price greater than the price of all the books belonging to Novel category can be specified as ? SELECT ISBN, Book_title, Price ? FROM BOOK ? WHERE Price &lt; ALL (SELECT Price ? FROM BOOK WHERE Category = 'Novel'); ? EXISTS operator evaluates to true if a subquery returns at least one tuple as a result otherwise, it returns false value. For example, the command to retrieve the details of publishers having at least one book published can be specified as ? SELECT P_ID, Pname, Address ? FROM PUBLISHER 130
? WHERE EXISTS (SELECT * ? FROM BOOK WHERE PUBLISHER.P_ID = BOOK.P_ID); On the other hand, NOT EXISTS operator evaluates to true if subquery returns no tuple as a result. For example, the command to retrieve the details of publishers having not publishing any book can be specified as SELECT P_ID, Pname, Address FROM PUBLISHER WHERE NOT EXISTS (SELECT * FROM BOOK WHERE PUBLISHER.P_ID = BOOK.P_ID); 5.5 ADDITIONAL FEATURES OF SQL Some of the advanced features of SQL that help in specifying complex constraints and queries efficiently are assertions and views. Assertion is a condition that must always be satisfied by the database. Assertions do not fall into any of the categories of constraints discussed earlier. But, view is a virtual relation which derives its data from one or more existing relations. The result of view is not physically
stored. The two standard features of SQL that are related to programming in database are stored procedures and triggers. The stored procedures are program modules that are stored by the DBMS at the database server. These procedures can be invoked and executed whenever required using the single SQL statement from various applications that have access to the database. Whereas, triggers are type of stored procedures which are automatically invoked and are needed to perform complex data verification operations.
This section discusses assertions, views, stored procedure, and triggers. 5.5.1 Assertions Some of the simple data integrities can be specified while defining the structure of the relation with the help of various constraints like PRIMARY KEY, NOT NULL, UNIQUE, CHECK, etc. In addition, the referential integrity can be specified using the FOREIGN KEY constraint. However, there are some data integrities, which cannot be specified using any of the constraints discussed so far. Such type of constraints can be specified using the assertions. The DBMS enforces the assertion on the database and the constraints specified in assertion must not be violated. Assertions are always checked whenever modifications are done in corresponding relation. The syntax to create an assertion can be specified as CREATE ASSERTION &gt;assertion_name&lt; CHECK (&gt;condition&lt;); The assertion name is used to identify the constraints specified by the assertion and can be used for modification and deletion of assertion, whenever required. An assertion is implemented by writing a query that retrieves any tuples that violates the specified condition. Then this query is placed inside a NOT EXISTS clause, which indicates that the result of this query must be empty. Hence, the assertion is violated whenever the result of this query is not empty. For example, the price of textbook must not be less than the minimum price of novel, the assertion for this requirement can be specified as CREATE ASSERTION price_constraint CHECK (NOT EXISTS (SELECT * FROM BOOK WHERE Category = 'Textbook' AND (Price&gt;(SELECT MIN (Price) FROM BOOK WHERE Category='Novel') 131
)); In this command, query retrieves tuples with category as Textbook and price less than the minimum price of book with Novel category. The result of this query must be empty; otherwise it will violate the assertion. DBMS tests the assertion for its validity when it is created. After its creation,
future modification to the database is allowed only if it does not

violate the assertion. Note that, the assertions should be used to specify complex constraints only that cannot be specified by any of the other constraints, as assertions introduce a significant amount of overhead in terms of time and cost. 5.5.2 Views A view is a virtual relation, whose contents are derived from already existing relations and it does not exist in physical form. The contents of view are determined by executing a query based on any relation and it does not form the part of database schema. Each time a view is referred to, its contents are derived from the relations on which it is based. A view can be used like any other relation, which is, it can be queried, inserted into, deleted from, and joined with other relations or views, though with some limitations on update operations. These are very useful in the situations where only parts of relations are to be accessed frequently instead of complete data. The CREATE VIEW command of SQL can be used for creating views. This command provides the name to the view and specifies the list of attributes and tuples to be included using a subquery. The syntax to create a view is given here. CREATE VIEW &gt;view_name&lt; AS &gt;subquery&lt;; For example, the command to create a view containing details of books which belong to Textbook and Language Book categories can be specified as CREATE VIEW BOOK_1 AS SELECT * FROM BOOK WHERE Category IN ('Textbook', 'Language Book'); This command creates a view, named as BOOK_1, having details of books satisfying the condition specified in WHERE clause. The view created like this consists of all the attributes of BOOK relation; however, only selected attributes can also be included in the view and they can be given another name also. For example, consider the command given here. CREATE VIEW BOOK_2(B_Code, B_Title, B_Category, B_Price) AS SELECT ISBN, Book_title, Category, Price FROM BOOK WHERE Category IN ('Textbook', 'Language Book'); This command creates a view BOOK_2, which consists of the attributes, ISBN, Book_title, Category, and Price from the relation BOOK with new names, namely, B_Code, B_Title, B_Category, and B_Price, respectively. Now queries can be performed on these views as they are performed on the other relations. For example, consider the commands given here. SELECT * FROM BOOK_1; SELECT * FROM BOOK_2 WHERE B_Price &lt; 30; SELECT B_Title, B_Category FROM BOOK_2 WHERE B_Price BETWEEN 30 AND 50; 132
Views can be based on more than one relation. For example, the command to create a view consisting of attributes Book_title, Category, Price and P_ID of BOOK relation, Pname and State of PUBLISHER relation can be specified as CREATE VIEW BOOK_3 AS SELECT Book_title, Category, Price, BOOK.P_ID, Pname, State FROM BOOK, PUBLISHER WHERE BOOK.P_ID = PUBLISHER.P_ID; The views that are based on more than one relation are said to be complex views. These types of views are inefficient as they are time consuming to execute, especially if multiple queries are involved in the view definition. Since their contents are not physically stored, they are executed each and every time they are referred to. As an alternative to this, certain database systems allow views to be physically stored, also known as materialized views. These types of views save the time spent to execute the subqueries each and every time they are referred to. Such views are updated whenever tuples are inserted, deleted, or modified in the underlying relations. The view is stored temporarily, that is, if view is not queried for a certain period of time, it is physically removed and is recomputed when it is again referred in future. While implementing the concept of materialized views, their advantages must be compared with the cost incurred for storing them and their updations. A view is updateable if it is based on single relation and the update query can be mapped on to the already existing relation successfully under certain conditions. Any view can be made non-updateable, by adding the WITH READ ONLY clause. That is, no INSERT, UPDATE, or DELETE command can be carried over that view. Views can also be created using the queries based on other views. For example, the command to create view based on the view BOOK_3, having details, where publishers belong to the state New York can be specified as CREATE VIEW BOOK_4 AS SELECT * FROM BOOK_3 WHERE State = 'New York'; Like relation, view can also be deleted from the database by using DROP VIEW command. For example, to delete the view BOOK_1 from the database, the command can be specified as DROP VIEW BOOK_1; 5.5.3 Stored Procedures Stored procedures are procedures or functions that are stored and executed by the DBMS at the database server machine. In SQL standard, stored procedures are termed as persistent stored modules (PSM), as these procedures, like data, are persistently stored by the DBMS. Stored procedures improve performance, as these procedures are compiled only at the time of their creation or modifications. Hence, whenever these procedures are called for the execution the time for compilation is saved. Stored procedures are beneficial when a procedure is required by different applications located at remote sites, as it is stored at server site and can be invoked by any of the applications. This eliminates duplication of efforts in writing SQL application logic and makes code maintenance easy. Most of the DBMSs allow stored procedures to be written in any general-purpose programming language. As an alternative, stored procedures may consist of simple SQL commands like INSERT, SELECT, UPDATE, etc. Stored procedures can be compiled and executed with different parameters and results. They can accept input parameters and pass values to output parameters. A stored procedure can be created as shown here. CREATE PROCEDURE &gt;name&lt; (&gt;parameters 1 , ..., parameters n &lt;) &gt;local_declarations&lt; &gt;body of procedure&lt;; Parameters and local declarations are optional and if declared must be of valid SQL data types. Parameters declared must have one of the three modes, namely, IN, OUT, or INOUT specified for it. Parameters specified with IN mode are 133

arguments passed to the stored procedure and act like a constant, whereas, OUT parameters act like an un-initialized variables and they cannot appear on the right hand side of = symbol. Parameters with INOUT mode have combined properties of both IN and OUT mode. They contain values to be passed to the stored procedure and also can be assigned values to be returned from the stored procedure. For example, the procedure to update the value of price of a book with a given ISBN of relation BOOK can be specified as CREATE PROCEDURE Update_price (IN B_ISBN VARCHAR(15), IN New_Price NUMERIC(6,2)) UPDATE BOOK SET Price = New_Price WHERE ISBN = B_ISBN; Functions are required when value is required to be returned to the calling program since procedures cannot return a value. The function can be created as shown here. CREATE FUNCTION &gt;name&lt; (&gt;parameters 1 , …, parameters n &lt;) RETURNS &gt;return_type&lt; &gt;local_declarations&lt; &gt;body of function&lt; ; For example, the function returning a rating of book with a given ISBN and author ID can be specified as CREATE FUNCTION Book_rating (IN B_ISBN VARCHAR(15), IN Au_ID VARCHAR(4)) RETURNS NUMERIC(2) DECLARE B_rating NUMERIC(2) SET B_rating=(SELECT Rating FROM REVIEW WHERE ISBN=B_ISBN AND R_ID = Au_ID); RETURN B_rating; SQL/PSM SQL/PSM is a part of SQL standard that includes programming constructs for writing the coding part of persistent stored modules. It also includes constructs for specifying conditional statements and looping statements. The conditional statements in SQL/PSM can be specified as IF &gt;condition&lt; THEN &gt;statements&lt; ELSEIF &gt;condition&lt; THEN &gt;statements&lt; : ELSEIF &gt;condition&lt; THEN &gt;statements&lt; ELSE &gt;statements&lt; END IF ; The while looping construct can be specified as WHILE &gt;condition&lt; DO &gt;statements&lt;; END WHILE ; The repeat looping construct can be specified as REPEAT &gt;statements&lt;; UNTIL &gt;condition&lt; 134
END REPEAT ; For example, a function that searches a book with a given ISBN and declares it to be of High, Medium, or Low price can be written as CREATE FUNCTION B_price (IN B_ISBN VARCHAR(15)) RETURNS VARCHAR(7) DECLARE Book_price NUMERIC(6,2) SET Book_price=(SELECT Price FROM BOOK WHERE ISBN=B_ISBN); IF Book_price&lt;100 THEN RETURN 'High' ELSEIF Book_price&lt;50 THEN RETURN 'Medium' ELSE RETURN 'Low' END IF ; Similarly, loops can also be used in the code of stored procedure. Moreover, loops can be named in the code. The name of loop can be used for breaking out of the loop based on some condition by using the statement LEAVE &gt;loop_name&lt;. If the procedure or function is written in some general-purpose programming language, it is necessary to specify the language and the file name where program code is stored. In such a case, the procedure can be created as shown here. CREATE PROCEDURE &gt;procedure_name&lt;(&gt;parameters&lt;) LANGUAGE &gt;name of programming language&lt; EXTERNAL NAME &gt;file path name&lt;; Once the stored procedures or functions are created, they can be called in any application for execution. The calling statement can be specified as CALL &gt;procedure or function name&lt; (&gt;arguments&lt;); For example, the statements for calling procedures and functions can be specified as CALL Update_price('003-456-433-6', 28); CALL Book_rating('003-456-533-8', 'A004'); CALL B_price('001-987-760-9'); 5.5.4 Triggers A trigger is a type of stored procedure that is executed automatically when some database related events like, insert, update, delete, etc., occur. In addition, unlike procedures, triggers do not accept any arguments. The main aim of triggers is to maintain data integrity and also one can design a trigger for recording information, which can be used for auditing purposes. The triggers are mainly needed for following purposes. ? Implementing and maintaining complex integrity constraints. ? Recording the changes for auditing purposes. ? Automatically passing signal to other programs that action needs to be taken whenever specific changes are made to a relation. Triggers are usually defined on relations. However, it can be defined on views and can also be used to execute other triggers, procedures, and functions. Although triggers like constraints are defined to maintain the database integrity, yet they are different from constraints in the following ways. ? Triggers affect only those tuples that are inserted after the trigger is enabled, whereas, constraints affect all tuples in the relation. 135

? Triggers allow enforcing the user-defined constraints, which are not possible through standard constraints supported by database. A database having triggers associated with it is known as active database. A trigger consists of three parts. ? Event: any change in the database that activates the trigger. ? Condition: when the trigger is activated, the condition is checked. ? Action: a procedure that is to be executed, whenever the trigger is activated and the corresponding condition is satisfied. In other words, a trigger is an event-condition-action rule that states that whenever a specified event occurs and the condition is satisfied, the corresponding action must be executed. The trigger can be created as shown here. CREATE TRIGGER &gt;trigger_name&lt; [BEFORE or AFTER] [INSERT or UPDATE or DELETE] ON &gt;relation_name&lt; [FOR EACH ROW] WHEN &gt;condition&lt; &gt;statements&lt;; For example, when value in Phone attribute of new inserted tuple of a relation AUTHOR is empty, indicating absence of phone number, the trigger to insert a null value in this attribute can be specified as CREATE TRIGGER Setnull_phone BEFORE INSERT ON AUTHOR REFERENCING NEW ROW AS nr FOR EACH ROW WHEN nr.Phone = ' ' SET nr.Phone = NULL; The FOR EACH ROW clause makes sure that trigger is executed for every single tuple processed. Such type of trigger is known as row level trigger. Whereas, the trigger, which is executed only once for specified statement, regardless of the number of tuples being effected as a result of that statement, is known as statement level trigger. The FOR EACH STATEMENT clause specifies the trigger as statement level trigger. For example, trigger defined for a INSERT command, will be executed only once, regardless of the number of tuples inserted through single INSERT statement. The statement level triggers are the default type of triggers, which are identified by omitting the FOR EACH ROW clause. Learn More Triggers are provided with a separate namespace from procedures, package, relations (sharing the same namespace), which means that triggers can have same name as of a relation or procedure. The REFERENCING NEW ROW AS clause can be used to create a variable for storing the new values of an updated tuple. Similarly, REFERENCING OLD ROW AS clause can be used to create variables for storing old values of an updated tuple. In addition, the clauses REFERENCING NEW TABLE AS or REFERENCING OLD TABLE AS can be used to refer to temporary relation consisting of all the affected tuples. Such relations can be used with only AFTER triggers and not BEFORE triggers. Consider another example, to create the trigger for changing value of Price attribute to a default value $60, if the value entered by the user exceeds the upper limit of $200 in case of insertion in relation BOOK. The trigger can be specified as CREATE TRIGGER New_price BEFORE INSERT ON BOOK FOR EACH ROW WHEN (new.Price&lt;200) BEGIN new.Price = 60; END 136
The keyword new and old can be used to refer to the values after and before the modifications are made. Triggers can be enabled and disabled by using the ALTER TRIGGER command. The triggers which are not required can be removed by using the DROP TRIGGER command. For example, consider the following statements. ALTER TRIGGER New_price DISABLE ; ALTER TRIGGER New_price ENABLE ; DROP TRIGGER New_price; The triggers come with lot of advantages. However, triggers must be used with great care as sometimes action of one trigger can lead to the activation of another trigger. Such triggers are known as recursive triggers. In the worst situation, it can lead to an infinite chain of triggering. A database system typically limits the length of such chains of triggers to 16 or 32 and considers longer chains of triggering an error. 5.6
ACCESSING DATABASES FROM APPLICATIONS So far, we have discussed how queries are executed in SQL. In most of the situations, databases are accessed through software programs implementing database applications. These types of software are usually developed in
general-purpose programming languages, such as C, Java, COBOL, Pascal, and FORTRAN.

And these general-purpose programming languages are known as host languages. Various techniques have been developed for accessing databases from other programs. Some of the techniques for using SQL statements in host language are embedded SQL, cursors, dynamic SQL, ODBC, JDBC, and SQLJ. These techniques are discussed in this section. 5.6.1 Embedded SQL For accessing a database from any application program, the SQL statements are embedded within an application program written in host language. The use of SQL statements within a host language program is known as embedded SQL. These statements are also known as static, that is, they do not change at runtime. SQL statements included in a host language program must be marked so that the pre-processor can handle them before invoking the compiler for a host language. The embedded SQL statement is prefixed with the keywords EXEC SQL to distinguish it from the other statements of host language. Prior to compilation, an embedded SQL program is processed by a special pre-processor or pre-compiler, which identifies the SQL statements during program scan, extracts and pass it to the DBMS for processing. The end of SQL statements is identified by encountering a semicolon (;) or a matching END-EXEC. For the discussion of embedded SQL, consider C as a host language. Variables of host language can be referred in SQL statements. Such variables are also known as host variables and are prefixed by a colon (:) when they appear in SQL statements and are declared between the commands EXEC SQL BEGIN DECLARE SECTION and EXEC SQL END DECLARE SECTION. The declaration of these variables is similar to the declaration of variables in C language and is separated by semicolons. These variables are also known as shared variables as they are shared by both C language and SQL statements. For example, the declaration of variables corresponding to the attributes of relation BOOK will be like as shown here. EXEC SQL BEGIN DECLARE SECTION; varchar c_ISBN[15], c_book_title[50], c_category[20]; char c_p_id[4]; int c_copyright_date[10], c_year, c_page_count; float c_price; int SQLCODE; char SQLSTATE[6]; EXEC SQL END DECLARE SECTION; Program variables may or may not have same names as that of attributes in a corresponding relation. The SQL data type NUMERIC can be mapped to C data type int or float (if includes decimal portion). The SQL fixed length and variable length strings can be mapped to arrays of characters of type char and varchar, respectively. The variables SQLCODE and SQLSTATE are used to communicate errors and exception conditions between the database system and the host language program. 137
The SQLCODE is an integer variable, which returns a positive value (like SQLCODE=100) when there is no more data in the resultant relation. The SQLCODE returns negative value when an error occurs. SQLSTATE is a five character code the 6 th character is for null character in C language. The value 00000 of the variable SQLSTATE indicates error free condition. It associates predefined values with several common error conditions. SQL statements can appear anywhere in the host language program, where other statements of host language can appear. Before executing any SQL statements, the host program must establish a connection with the database as shown here. EXEC SQL CONNECT TO &gt;server_name&lt; AS &gt;connection_name&lt; AUTHORIZATION &gt;user_name and password&lt;; In this command, server_name identifies the server to which a connection is to be established and the connection_name is the name provided to the connection. In addition, user_name and password identifies the authorized user. While programming, more than one connection can be established with only one connection active at a time. When a connection is no longer needed, it can be ended by using the command as shown here. EXEC SQL DISCONNECT &gt;connection_name&lt;; Note that the commands are terminated by semicolon. Assuming that the required connection is already established, the command to insert a tuple in relation using host variables can be specified as EXEC SQL INSERT INTO BOOK VALUES (:c_ISBN, :c_book_title, :c_category, :c_price, :c_copyright_date, :c_year, :c_page_count, :c_p_id); Consider another example to retrieve the details of book with ISBN value stored in the variable c_ISBN, the embedded SQL statement can be specified as EXEC SQL SELECT Book_title, Category, Price, Year, Page_count INTO :c_book_title, :c_category, :c_price, :c_year, :c_page_count FROM BOOK WHERE ISBN=:c_ISBN; In this command, the details of book satisfying the condition is stored in host variables. As a result, only single tuple is retrieved and hence, can be stored in the host variables. However, while programming, more than one tuple satisfying the condition can also be retrieved. To deal with such a situation, the concept of cursors can be used, which is discussed in the following section. 5.6.2 Cursors Cursor is a mechanism that provides a way to retrieve multiple tuples from a relation and then process each tuple individually in a host program. The cursor is first opened, then processed, and then closed. Cursor can be declared on any relation or any SQL statement that returns a set of tuples. Once the cursor is declared, it can be opened, moved to n th tuple and closed. When the cursor is opened, it fetches the query result from the database and it is positioned just before the first tuple. In the query result, any individual tuple of a relation can be fetched by pointing cursor to the desired tuple. Sometimes the cursor is opened implicitly (automatically) by the RDBMS especially for the queries returning single tuple. However, for the queries returning more than one tuple, explicit cursors (defined by user) are declared. In general, to use the explicit cursor, the following steps are performed. 1. Declare the cursor: The cursor is declared using the DECLARE command. 2. Open the cursor: Before using the cursor, it must be opened after it has been declared. 3. Fetch tuple from the cursor: After opening the cursor, the tuples associated with the cursor can be processed individually with the help of FETCH command. 138

4. Close the cursor: When cursor is not required, it must be closed by using the CLOSE command. The declaration of cursor takes the following form. DECLARE &gt;cursor_name&lt; [INSENSITIVE][SCROLL] CURSOR [WITH HOLD] FOR &gt;query&lt; [ORDER BY &gt;attribute_list&lt;] [FOR READ ONLY | FOR UPDATE [OF &gt;attribute_list&lt;]]; The clause FOR UPDATE OF is added in the declaration of cursor, indicating that the tuples associated with cursor will be retrieved for updating and the attribute to be updated is specified with it. The ORDER BY clause is used to order the tuples in the resultant relation. A cursor can be declared as read only cursor using the clause FOR READ ONLY, indicating the tuples retrieved as a result of SQL query cannot be modified. Other keywords that can be used in the declaration of cursor are SCROLL, INSENSITIVE, and WITH HOLD. When the keyword SCROLL is specified, then that cursor is scrollable, this means that the FETCH command can be used to position the cursor in a flexible manner instead of default sequential access. The other two keywords INSENSITIVE and WITH HOLD are used to refer the transaction characteristics of database programs. For example, the program segment to increment the price of books belonging to a given category from relation BOOK by the value entered by the user can be written as shown in Figure 5.1. When the cursor is opened, it is positioned at the beginning of the first tuple. When FETCH command is executed, the attribute values in the corresponding tuple are read and stored in respective host variables in the specified order. To read consecutive tuples, the FETCH command is executed repeatedly. If the execution of FETCH command results in the moving of cursor to the end of last tuple, a positive value (SQLCODE&lt;0) is returned in SQLCODE. Positive value in SQLCODE indicates presence of no more data in the resultant relation. This state of SQLCODE is used to terminate the loop. The WHERE CURRENT OF Cur1 clause in the embedded UPDATE command is used to specify that the tuple referred currently by the cursor is the tuple to be updated. After the complete processing of resultant relation, the cursor is closed using CLOSE command. Things to Remember Once the cursor is opened it cannot be reopened. That is, cursor must be closed before reopening it. Also, the cursor always moves forward and not backwards. 5.6.3 Dynamic SQL Embedded SQL allows the integration of SQL statements within the host language program. Such statements are of static nature, that is, once these statements are written in the program then they cannot be modified at any time. Hence, to specify a new query, new program must be written to accommodate the new query. Dynamic SQL, unlike embedded SQL statements, are generated at the run-time and it is placed as a string in the host variable. The SQL statements created like this are passed to the DBMS for further processing. Dynamic SQL is slower than embedded SQL as it involves time to generate a query also during the runtime. However, it is more powerful than embedded SQL, as queries can be generated at runtime as per varying user requirements. printf("Enter the category of book : "); scanf("%s", c_category); EXEC SQL DECLARE Cur1 CURSOR FOR SELECT Book_title, Price, Year FROM BOOK WHERE Category=:c_category ORDER BY Book_title FOR UPDATE OF Price; EXEC SQL OPEN Cur1; 139
EXEC SQL FETCH FROM Cur1 INTO :c_Book_title, :c_Price, :c_Year; WHILE(SQLCODE ==0) { printf("Enter the increment amount : "); scanf("%f", &inc); EXEC SQL UPDATE BOOK SET Price = Price + :inc WHERE CURRENT OF Cur1; EXEC SQL FETCH FROM Cur1 INTO :c_Book_title, :c_Price, :c_Year; } EXEC SQL CLOSE Cur1; Fig. 5.1 Retrieving tuples using cursor For example, consider program segment given in Figure 5.2. This sample program allows user to enter the update SQL query to be executed. EXEC SQL BEGIN DECLARE SECTION; char sqlquerystring[200]; EXEC SQL END DECLARE SECTION; printf("Enter the required update query : \
n");
scanf("%s", sqlquerystring); EXEC SQL PREPARE sqlcommand FROM :sqlquerystring; EXEC SQL EXECUTE sqlcommand; Fig. 5.2 Program segment using dynamic SQL In this code segment, first the string sqlquerstring is declared, which is used to hold the SQL query statement entered by the user. After prompting the required query from the user, it is converted into corresponding SQL command by using the PREPARE command. This query is executed by using the EXECUTE command. The query entered by the user can be in the form of a single string or it can be created by using concatenation of strings. 5.6.4 Open Database Connectivity (ODBC) To access the database from general-purpose programming language, the application programs needs to set up a connection with the database server. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API) that the application programs can use to establish a connection with the database. Once the connection is established, the application program can communicate with the database through queries. Most DBMS vendors provide ODBC drivers for their systems. An application program from the client-site can access several DBMSs by making ODBC API call. The requests from the client program are then processed at the server-sites and the results are sent back to the client program. Consider an example of C code using ODBC API given in Figure 5.3. void Example() { HENV En1; //environment HDBC Cn; //to establish connection RETCODE Err; SQLAllocEnv(&En1); SQLAllocConnect(En1,&Cn); SQLConnect(Cn, "db.onlinebook.edu", SQL_NTS, "John", SQL_NTS, "Passwd", SQL_NTS); int c_Price1, c_Price2; int L1, L2; 140

HSTMT st; char *Query = “SELECT MAX(Price), MIN(Price) FROM BOOK GROUP BY Category”; SQLAllocStmt(Cn, &st); Err = SQLExecDirect(st, Query, SQL_NTS); if (Err == SQL_SUCCESS) { SQLBindCol(st, 1, SQL_C_INT, &c_Price1, 0, &L1); SQLBindCol(st, 2, SQL_C_INT, &c_Price2, 0, &L2); while(SQLFetch(st) == SQL_SUCCESS) { printf(“Maximum Price is %d”, c_Price1); printf(“Minimum Price is %d”, c_Price2); } } SQLFreeStmt(st, SQL_DROP); SQLDisconnect(Cn); SQLFreeConnect(Cn); SQLFreeEnv(En1); } Fig. 5.3 An example of ODBC code In the beginning of the program, the variables En1, Cn, and Err of types HENV, HDBC, and RETCODE, respectively are declared. The variable En1 and Cn are used to allocate an SQL environment and database connection handle, respectively. The variable Err is used for error detection. Further, the program establishes a connection with the database by using SQLConnect() function, which accepts parameters, database connection handle (Cn), server (db.onlinebook.edu), user identifier (John), and the password (Passwd). Notice the use of constant SQL_NTS, which denotes that the previous argument is a null-terminated string. After establishing a connection, the program communicates with the database by sending a query to SQLExecDirect() function. The attributes of the query result are bounded to corresponding C variables by using SQLBindCol() function. The parameters passed to the SQLBindCol() are: ? First parameter (st): stores the result of the query ? Second parameter (1 or 2): determines the location of an attribute in the result of a query ? Third parameter (SQL_C_INT): specifies the required data type conversion of an attribute from SQL to C. ? Fourth parameter (&c_Price1 or &c_Price2): specifies the address of the C variable where the attribute value is to be stored. Note that the values of last two parameters passed to SQLBindCol() function, depends on the data type of an attribute. For example, in case of fixed-length types, such as float or integer the fifth parameter is ignored and negative value in last parameter indicates null value in an attribute. When the resultant tuple is fetched using SQLFetch() function, the attribute values of the query are stored in corresponding C variables. The SQLFetch() function executes the statement st as long as it returns the value SQL_SUCCESS. In each iteration of while loop, the attribute values for each category are stored in corresponding C variables and are displayed through printf statements. The connection must be closed when it is no more required, using SQLDisconnect() function. Also, all the resources that are allocated must be freed. 5.6.5 Java Database Connectivity (JDBC) 141

Accessing a database in Java requires Java Database Connectivity (JDBC). JDBC provides a standard API that is used to access databases, through Java, regardless of the DBMS. All the direct interactions with specific DBMSs are accomplished by DBMS specific driver. The four main components required for the implementation of JDBC are: application, driver manager, data source specific drivers, and corresponding data sources. An application establishes and terminates the connection with a data source. The main goal of driver manager is to load JDBC drivers and pass JDBC function calls from the application to the corresponding driver. The driver establishes the connection with data source. The driver performs various basic functions like submitting requests and returning results. In addition, the driver translates data, error formats, and error codes from a form that is specific to data source into the JDBC standard. The data source processes commands from the driver and returns the results. For understanding the connectivity of Java program with JDBC, consider sample program segment given in Figure 5.4. In this program segment, following steps are taken when writing a Java application program accessing database through JDBC function calls. 1. Appropriate drivers for the database are loaded by using Class.forName. 2. The getConnection() function of DriverManager class of JDBC is used to create the connection object. The first parameter (URL) specifies the machine name (db.onlinebook.edu) 3. public static void Sample(String DB_id, String U_id, String Pword) 4. { 5. String URL = ”jdbc:oracle:oci8:@db.onlinebook.edu:100:onbook_db”; 6. Class.forName(“oracle.jdbc.driver.OracleDriver”); 7. Connection Cn=DriverManager.getConnection(URL,U_id, Pword); 8. Statement st = Cn.createStatement(); 9. try 10. { 11. st.executeUpdate(“INSERT INTO AUTHOR VALUES(‘A010’, ‘Smith 12. Jones’, ‘Texas’)”); 13. } 14. catch(SQLException se) 15. { 16. System.out.println(“Tuple cannot be inserted.”+se); 17. } 18. ResultSet rs = st.executeQuery(“SELECT Aname, State from AUTHOR 19. WHERE City = ‘Seatle’”); 20. while(rs.next()) 21. { 22. System.out.println(rs.getString(1) + “ ”+ rs.getString(2)); 23. } 24. st.close(); 25. Cn.close(); } Fig. 5.4 An example of JDBC code where the server runs, the port number (100) used for communication, schema (onbook_db) to be used and the protocol (jdbc:oracle:oci8) used to communicate with the database. To connect to the database, username and password are also required which are specified by the strings U_id and Pword, respectively, the other two parameters of getConnection() function. 142

26. A statement handle is created for the connection established which is used to execute an SQL statement. In this example, st.executeUpdate is used to execute an INSERT statement of SQL. The try {..} catch{..} is used to catch any exceptions that may arise as a result of executing this query statement. 27. Another query is executed using the statement st.executeQuery. The result of this may consist of set of tuples, which is assigned to rs of type ResultSet. 28. The next() function is used to fetch one tuple at a time from the result set rs. The value of attributes of a tuple is retrieved by using the position of the attribute. The attribute Aname is at first (1) position and State is at second (2) position. The values for the attributes can also be retrieved by using its name as shown here. 29.
System.out.println(rs.getString(“Aname”)+ “ ” +rs.getString(“State”)); 30. The connection must be closed after it is no more required at the end of procedure. The special type of statement also known as prepared statement, can be used to specify SQL query, in which unknown values are replaced by ‘?’. The user provides the unknown values later at the run-time. Some database systems compile the query when it is prepared and whenever the query is executed with new values, database uses this compiled form of this query. The setString() function is used to specify the values for the parameters. Consider the following statements, to understand the concept of prepared statement. PreparedStatement ps = Cn.prepareStatement(“INSERT INTO BOOK VALUES(?,?,?)”); ps.setString(1, “A010”); //assigns value to first attribute ps.setString(2, “Smith Jones”); //assigns value to second //attribute ps.setString(3, “Texas”); //assigns value to third attribute ps.executeUpdate(); The values can be assigned to the corresponding attributes through variables also instead of using literal values. Prepared statements are preferred in the situations when query uses the values provided by the user. JDBC also provides a CallableStatement interface, which can be used for invoking SQL stored procedures and functions. CallableStatement is a subclass of PreparedStatement. For example, the command to call the procedure, say Number_Of_Books, can be specified as CallableStatement cs1 = Cn.prepareCall(“{call Number_Of_Books}”); ResultSet rs = cs1.executeQuery(); A stored procedure may contain multiple SQL statements or a series of SQL statements, thus, resulting into many different ResultSet objects. In this example, it is assumed that there is only single ResultSet. 5.6.6 SQL-Java (SQLJ) SQLJ is a standard that has been used for embedding SQL statements in Java programming language, which is an object- oriented language like Java. In SQLJ, a pre-processor called “SQLJ translator” converts SQL statements into Java, which can be executed through JDBC interface. Hence, it is essential to install a JDBC driver while using SQLJ. When writing SQLJ applications, regular Java code is written and SQL statements are embedded into it using a set of rules. SQLJ applications are pre-processed using SQLJ translation program that replaces the embedded SQLJ code with calls to an SQLJ library. Any Java compiler can then compile this modified program code. The SQLJ library calls the corresponding driver, which establishes the connection with the database system. The use of SQLJ improves the productivity and manageability of JAVA code as code becomes compact and no runtime syntax errors occur as SQL statements are checked at compile time. Moreover, it allows sharing of Java variables with SQL statements. For understanding the coding in SQLJ, consider a sample SQLJ program segment given in Figure 5.5. This program retrieves the book details belonging to a given category. 143
String Book_title; float Price; String Category; #sql iterator Bk(String Book_title, float Price); Bk book = {SELECT Book_title, Price INTO :Book_title, :Price FROM BOOK WHERE Category = :Category}; while(book.next()) { System.out.println(book.Book_title() + “, ” + book.Price()); } book.close(); Fig. 5.5 An example of SQLJ code SQLJ statements always begin with the #sql keyword. The result of SQL queries is retrieved through the objects of iterator, which are basically a type of cursor. The iterator is associated with the tuples and attributes appearing in the query result. An iterator is declared as an instance of iterator class. The usage of an iterator in SQLJ basically goes through following four steps. 1. Declaration of iterator class: In the Figure 5.5, the iterator class Bk is declared by using the statement. #sql iterator Bk(String Book_title, float Price); 2. Creation and initialization of iterator object: An object of iterator class is created and initialized with a set of tuples returned as a result of SQL query statement. 3. Accessing the tuples with the help of iterator object: Iterator is set to the position just before the first row in the result of query, which becomes the current tuple for the iterator. The next() function is then applied on the iterator repeatedly to retrieve the subsequent tuples from the result. 4. Closing the iterator object: An object of iterator is closed after all the tuples associated with the result of SQL query are processed. There are two types of iterator classes, namely, named and positional iterators. In case of named iterators both the variable types and the name of each column of the iterator is specified. This helps in retrieving the individual columns by their name. Like, in the Figure 5.5, the sample program to retrieve Book_title from the iterator book, the expression book.Book_title() is used. While, in case of positional iterators, only the variable type for each column of iterator is specified. The individual columns of the iterator are accessed using the FETCH. .INTO statement like embedded SQL. Both types of iterators have same performance and can be used according to the user requirement. NOTE SQLJ is much easier to read than JDBC code. Thus, SQLJ reduces software development and maintenance cost. SUMMARY 1. The structured query language (SQL) pronounced as “ess-que-el”, is a language which can be used for retrieval and management of data stored in relational database. It is a non-procedural language as it specifies what is to be retrieved rather than how to retrieve it. 2.
Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. 3.

SQL provides different types of commands, which can be used for different purposes. These commands are divided into two major categories, namely, data definition language (DDL) and data manipulation language (DML). 4.
Data definition language (DDL) commands are used for defining relation schemas, deleting relations, creating indexes, and modifying relation schemas. 5.
Present day database systems provide a three level hierarchy for naming different objects of relational database. This hierarchy comprises catalogs, which in turn consists of schemas, and various database objects are incorporated within the schema. 144
6. SQL supports various data types such as numeric, integer, character, character varying, date and time, Boolean, and timestamp. 7. In addition to built-in data types, user-defined data types can also be created. The user-defined data type can be created using the CREATE DOMAIN command. 8. The CREATE TABLE command is used to define a new relation, its attributes, and its data types. In addition, various constraints such as key, integrity, and referential constraints along with the restrictions on attribute domains and null values can also be specified within the definition of a relation. 9. ALTER TABLE command is used to change the structure of a relation. The user can add, modify, delete, or rename an attribute. The user can also add or delete a constraint. 10. The DROP TABLE command is used to remove an already existing relation, which is no more required as a part of a database. 11. DML commands are used for retrieving and manipulating data stored in the database. The basic retrieval and manipulating commands are SELECT, INSERT, UPDATE, and DELETE. 12. The criteria to determine the tuples to be retrieved is specified by using the WHERE clause. 13. A tuple variable is associated with a particular relation and is defined in the FROM clause. 14. SQL provides LIKE operator, which allows to specify comparison conditions on only parts of the strings. Different patterns are specified by using two special wildcard character, namely, per cent (%) and underscore (_). 15. There are various set operations that can be performed on relations, like, union, intersection, and difference. In SQL, these operations are implemented using UNION, INTERSECT, and MINUS operations, respectively. 16. SQL provides the ORDER BY clause to arrange the tuples of relation in some particular order. The tuples can be arranged on the basis of the values of one or more attributes. 17. The INSERT command is used to add a single tuple at a time in a relation. 18. The UPDATE command is used to make changes in the values of the attributes of the relations. The DELETE command is used to remove the tuples from the relation, which are no more required as a part of a relation. Tuples can be deleted from only one relation at a time. 19. SQL provides five built-in aggregate functions, namely, SUM, AVG, MIN, MAX, and COUNT. 20. The GROUP BY clause can be used in SELECT command to divide the relation into groups on the basis of values of one or more attributes. Conditions can be placed on the groups using HAVING clause. 21. A query that combines the tuples from two or more relations is known as join query. In such type of queries, more than one relation is listed in the FROM clause. 22. The query defined in the WHERE clause of another query is known as nested query or subquery. The query in which another query is nested is known as enclosing query. 23. Some of the advanced features of SQL that help in specifying complex constraints and queries efficiently are assertions and views. 24. Assertion is a condition that must always be satisfied by the database. 25. View is a virtual relation, whose contents are derived from already existing relations and it does not exist in physical form. Certain database systems allow views to be physically stored, also known as materialized views. 26.
The two standard features of SQL that are related to programming in database are stored procedures and triggers. 27. Stored procedures are procedures or functions that are stored
and executed by the DBMS at the database server machine. 28. A trigger is a type of stored procedure that is executed automatically when some database related events like, insert, update, delete, etc., occur. 29. Various techniques have been developed for accessing databases from other programs. Some of the techniques for using SQL statements in host language are embedded SQL, cursors, dynamic SQL, ODBC, JDBC, and SQLJ. 30. The use of SQL statements within a host language is known as embedded SQL. The embedded SQL statement is prefixed with the keywords EXEC SQL to distinguish it from the other statements of host language. 31. Cursor is a mechanism that provides a way to retrieve multiple tuples from a relation and then process each tuple individually in a host program. The cursor is first opened, then processed, and then closed. 145
32. Dynamic SQL, unlike embedded SQL statements, are generated at the run-time and it is placed as a string in the host variable. 33. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API) that the application programs can use to establish a connection with the database. 34. Accessing a database in Java requires Java Database Connectivity (JDBC). JDBC provides a standard API that is used to access databases, through Java, regardless of the DBMS. 35. SQLJ is a standard that has been used for embedding SQL statements in Java programming language, which is object-oriented language like Java. In SQLJ, a pre-processor called SQLJ translator converts SQL statements into Java, which can be executed through JDBC interface. 146
CHAPTER 6 RELATIONAL DATABASE DESIGN After reading this chapter, the reader will understand: ? Features of

a relational database to qualify as a good relational database ? Decomposition that resolves the problem of redundancy and null values ? The concept of functional dependencies that plays a key role in developing a good database schema ? Introduction of normal forms and the process of normalization ? Normal forms based on functional dependencies that are 1NF, 2NF, 3NF, BCNF ? Additional properties of decomposition, namely, lossless-join and dependency preservation property. ? Comparison of 3NF and BCNF ? Higher normal forms 4NF and 5NF that are based on multivalued and join dependencies ? The process of denormalization The primary issue in designing any database is to decide the suitable logical structure for the data, that is, what relations should exist and what attributes should they have. For a smaller application, it is feasible for the designer to decide directly the appropriate logical structure for the data. However, for a large application where the database gets more complex, to determine the quality or goodness of a database, some formal measures or guidelines must be developed. Normalization theory provides the guidelines to assess the quality of a database in designing process. The normalization is the application of set of simple rules called normal forms to the relation schema. In order

66%	MATCHING BLOCK 110/319	W

to determine whether a relation schema is in desirable normal form, information about the real world entity that

is modeled in a database is required. Some of this information exists in E-R diagram. The additional information is given by constraints like functional dependencies and other types of data dependencies (multivalued dependencies and join dependencies). This chapter begins with a discussion of various features of good relational database. It then introduces the concept of functional dependency, which plays a key role in designing the database. Finally, it discusses various normal forms on the basis of functional dependencies and other data dependencies. 6.1 FEATURES OF GOOD RELATIONAL DESIGN Designing of relation schemas is based on the conceptual design. The relation schemas can be generated directly using conceptual data model such as ER or enhanced ER (EER) or some other conceptual data model. The goodness of the resulting set of schemas depends on the designing of the data model. All the relations that are obtained from data model have basic properties, that is, relations having no duplicate tuples, tuples having no particular ordering associated with them and each element in the relation being atomic. However, to qualify as a good relational database design, it should also have following features. Minimum Redundancy A relational database should be designed in such a way that it should have minimum redundancy. Redundancy refers to the storage of same data in more than one location. That is, the same data is repeated in multiple tuples in the same relation or multiple relations. The main disadvantage of redundancy is the wastage of storage space. For example, consider the modified form of Online Book database in which information regarding books and publishers is kept in same relation, say, BOOK_PUBLISHER which is defined on schema BOOK_PUBLISHER(ISBN, Book_title, P_ID, Pname, Phone). The primary key for this relation is ISBN. Some tuples of the relation BOOK_PUBLISHER are shown in Figure 6.1. 147

Fig. 6.1 Instance of relation BOOK_PUBLISHER In this relation, there is repetition of names and phone numbers of publishers with all the books they have published. Now, assume that 100 books are published by any one publisher, and then all the information related to that publisher will be repeated unnecessarily over 100 tuples. In addition, due to redundancy, certain update anomalies can arise, which are as follows: ? Insertion anomaly: It leads to a situation in which certain information cannot be inserted in a relation unless some other information is stored. For example, in BOOK_PUBLISHER relation, information of a new publisher cannot be inserted unless he has not published any book. Similarly, information of a new book cannot be inserted unless information regarding publisher is not known. ? Deletion anomaly: It leads to a situation in which deletion of data representing certain information results in losing data representing some other information that is associated with it. For example, if the tuple with a given ISBN, say, 003-456-433-6 is deleted, the only information about the publisher P004 associated with that ISBN is also lost. ? Modification anomaly: It leads to a situation in which repeated data changed at one place results in inconsistency unless the same data is also changed at other places. For example, change in the name of a publisher, say, Hills Publications to Hills Publishers Pvt. Ltd. needs modification in multiple tuples. If the name of a publisher is not modified at all places, same P_ID will show two different values for publisher name in different tuples. NOTE Modification anomaly cannot occur in case of a single relation as a single SQL query updates the entire relation. However, it can be a serious problem, if the data is repeated in multiple relations. Fewer Null Values in Tuples Sometimes, there may be a possibility that some attributes do not apply to all tuples in a relation. That is, values of all the attributes of a tuple are not known (as discussed in insertion anomaly). In that case, null value can be stored in those tuples. For example, in relation BOOK_PUBLISHER, detail of a new book without having information regarding publisher can be inserted by placing null values in all the attributes of publisher. However, null values do not provide complete solution as they cannot be inserted in the primary key attributes and the attributes for which not null constraint is specified. For example, detail of a new publisher who has not published any book cannot be inserted, as null value cannot be inserted in the attribute ISBN since it is a primary key. Moreover, null values waste storage space and may lead to some other problems such as interpreting them, accounting for them while applying aggregate functions such as COUNT or AVG, etc. In a good database, the attributes whose values may frequently be null are avoided. If null values are unavoidable, they are applied in exceptional cases only. Thus, all the problems related to null values are minimized in a good database. 6.2 DECOMPOSITION While designing a relational database schema, the problems that are confronted due to redundancy and null values can be resolved by decomposition. In decomposition, a relation is replaced with a collection of smaller relations with specific 148 relationship between them. Formally, the decomposition of a relation schema R is defined as its replacement by a set of relation schemas such that

95%	MATCHING BLOCK 112/319	W

each relation schema contains a subset of the attributes of R.

For example, the relation schema BOOK_PUBLISHER represents a bad design as it permits redundancy and null values, hence, it is undesirable. Thus, the schema of this relation requires to be modified so that it has less redundancy of information and fewer null values. The undesirable features from the relation schema BOOK_PUBLISHER can be eliminated by decomposing it into two relation schemas as given here. BOOK_DETAIL(ISBN, Book_title, P_ID) and PUBLISHER_DETAIL(P_ID, Pname, Phone) The instances corresponding to the relation schemas BOOK_DETAIL and PUBLISHER_DETAIL are shown in Figure 6.2. Fig. 6.2 Decomposition From Figure 6.2, it is apparent that decomposition is actually a process of projection on a relation. Here, both the relations BOOK_DETAIL and PUBLISHER_DETAIL are the projections of original relation BOOK_PUBLISHER. Now, a tuple for a new publisher can be inserted easily in relation PUBLISHER_DETAIL, even if no book is associated with that particular P_ID in relation BOOK_DETAIL. Similarly, a tuple for a particular P_ID in relation PUBLISHER_DETAIL can be deleted without losing any information (taking foreign key constraint into account). Moreover, name or phone for a particular P_ID can be changed by updating a single tuple in relation PUBLISHER_DETAIL. This is more efficient than updating several tuples and the scope for inconsistency is also eliminated. During decomposition, it must be ensured

64%	MATCHING BLOCK 114/319	SA	BOOK SIZE.docx (D50092775)

that each attribute in R must appear in at least one relation schema R i so that no attributes are lost. Formally, ∪ R i = R This is called attribute preservation property of decomposition.

NOTE There are two additional properties of decomposition, namely, lossless-join and dependency preservation property which are discussed in Section 6.5. 149

For smaller applications, it may be feasible for a database designer to decompose a relation schema directly. However, in case of large real-world databases that are usually highly complex, such a direct decomposition process is difficult as there can be different ways in which a relational schema can be decomposed. Further, in some cases, decomposition may not be required. Thus, in order to determine whether a relation schema should be decomposed and which relation schema may be better than another, some guidelines or formal methodology is needed. To provide such guidelines, several normal forms have been defined. These normal forms are based on the notion of functional dependencies that are one of the most important concepts in relational schema design. 6.3 FUNCTIONAL DEPENDENCIES Functional dependencies are the special forms of integrity constraints that generalize the concept of keys. A given set of functional dependencies is used in designing a relational database in which most of the undesirable properties are not present. Thus, functional dependencies play a key role in developing a good database schema. Definition: Let X and Y be the arbitrary subsets of set of attributes of a relation schema R, then an instance r of R satisfies functional dependency (FD) X → Y, if and only if

100%	MATCHING BLOCK 115/319	SA	Sem III_ BCA_B21CA05DC.docx (D165628090)

for any two tuples t 1 and t 2 in r that have t 1 [X] = t 2 [X], they must also have t 1 [Y] = t 2 [Y].

That means,

83%	MATCHING BLOCK 116/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

whenever two tuples of r agree on their X value, they must

also agree on their Y value. Less formally, Y is functionally dependent on X, if and only if each X value in r is associated with it precisely one Y value. The statement X → Y is read as Y is functionally dependent on X or X determines Y. The left and right sides of a functional dependency are called determinant and dependent, respectively. Things to Remember The definition of FD does not require the set X to be minimal; the additional minimality condition must be satisfied for X to be a primary key. That is, in an FD X → Y that holds on R, X represents a key if X is minimal and Y represents the set of all other attributes in the relation. Thus, if there exists some subset Z of X such that Z → Y, then X is a superkey. Consider the relation schema BOOK_REVIEW_DETAIL(ISBN, Price, Page_count, R_ID, City, Rating) that represents review details of the books by the reviewers. The primary key of this relation is the combination of ISBN and R_ID. Note that the same book can be reviewed by different reviewers and same reviewer can review different books. The instance of this relation schema is shown in Figure 6.3. Fig. 6.3 An instance of BOOK_REVIEW_DETAIL relation 150

In this relation, the attribute Price is functionally dependent on the attribute ISBN (ISBN → Price), as tuples having same value for the ISBN have the same value for the Price. For example, the tuples having ISBN 001-987-760-9 have the same value 25 for Price. Similarly, the tuples having ISBN 002-678-980-4 have the same value 35 for Price. Other FDs such as ISBN → Page_count, Page_count → Price, R_ID → City are also satisfied. In addition, one more functional dependency {ISBN,R_ID} → Rating is also satisfied. That is, the attribute Rating is functionally dependent on composite attribute {ISBN,R_ID}. While designing database, we are mainly concerned with the functional dependencies that must hold on relation schema R. That is, the set of FDs which are satisfied by all the possible instances that a relation on relation schema R may have. For example, in relation BOOK_REVIEW_DETAIL, the functional dependency Page_count → Price is satisfied. However, in real- world, the price of two different books having same number of pages can be different. So, it may be possible that at some time in an instance of relation BOOK_REVIEW_DETAIL, FD Page_count → Price is not satisfied. For this reason, Page_count → Price cannot be included in the set of FDs that hold on BOOK_REVIEW_DETAIL schema. Thus, a particular relation at any point in time may satisfy some FD, but it is not necessary that FD always hold on that relation schema. The FDs can be represented conveniently by using FD diagram. In FD diagram, the attributes are represented as rectangular box. The arrow out of attribute depicts that on this attribute, the attributes to which arrow points are functionally dependent. The FD diagram for relation schema BOOK_REVIEW_DETAIL is shown in Figure 6.4. Fig. 6.4 FD diagram for BOOK_REVIEW_DETAIL As stated earlier, while designing a database, the FDs that must hold on the relational schemas are specified. However, for a given relation schema, the complete set of FDs can be very large. For example, once again, consider the relation schema BOOK_REVIEW_DETAIL. In addition to the functional dependencies discussed earlier, some other FDs that hold on this relation schema are {R_ID, ISBN} → City {R_ID, ISBN} → {City, Rating} {R_ID, ISBN} → ISBN {R_ID, ISBN} → {R_ID, ISBN, City, Rating} This large set of FDs can reduce the efficiency of database system. As each FD represents certain integrity constraint, the database system must ensure that any of the FDs should not be violated while performing any operation on the database. To reduce the effort spent in checking for violations, there should be some way to reduce the size of that set of FDs. That means for a particular set of FDs, say F 1 , it is required to find some other set, say F 2 , which is smaller than F 1 and every FD in F 1 is implied by FDs in F 2 . 6.3.1 Trivial and Non-Trivial Dependencies A dependency is said to be

52%	MATCHING BLOCK 118/319	SA	BOOK SIZE.docx (D50092775)

a trivial dependency if it is satisfied by all relations. For example, X → X is satisfied by all relations having attribute

X. Similarly, XY → X is satisfied by all relations having attribute X. Generally, an FD A → B is trivial, if and only if B ⊆ A. Consider BOOK_REVIEW_DETAIL relation (see Figure 6.3). In this relation, the FDs like ISBN → ISBN and {R_ID, ISBN} → ISBN are trivial dependencies. The dependencies, which are not trivial, are non-trivial dependencies. These are the dependencies that correspond to the essential integrity constraints. Thus, in practice, only non-trivial dependencies are considered. However, while dealing with the formal theory, all dependencies, trivial as well as non-trivial dependencies, have to be accounted. 151
6.3.2

76%	MATCHING BLOCK 117/319	W

Closure of a Set of Functional Dependencies For a given set F of functional dependencies,

some other dependencies also hold. That is, some functional dependencies are implied by F. For example, consider a relation schema R(A, B, C), such that both the FDs A → B and B → C hold for R. Then, FD A → C also holds on R as C is dependent on A transitively, via B. Thus, it can be said that FD A → C is implied.

87%	MATCHING BLOCK 119/319	SA	MCSDSC-2.2 Relational database Management syst ... (D151484310)

The set of all FDs that are implied by a given set F of FDs is called the closure of F, denoted as F+.

For a given F (if it is small), F + can be computed directly. However, if F is large, then some rules or algorithms are needed to compute F + . Armstrong proposed a set of inference rules or axioms (also known as Armstrong's axioms) to find logically implied functional dependencies. These rules are applied repeatedly for a given set F of functional dependencies, so that all of F + can be inferred. Let A, B, C, and D be the arbitrary subsets of the set of attributes of a relation schema R and A ∪ B is denoted by AB, then these axioms can be stated as ? Reflexivity rule: If

		SA	
	MATCHING BLOCK 120/319		248E1130_RDBMS.docx (D165247738)

B ⊆ A, then A → B holds. ? Augmentation rule: If A → B holds, then AC → BC holds. ? Transitivity rule: If both A → B and B → C hold, then A → C holds.

The axioms are complete as for a given set F of FDs, all FDs implied by F can be inferred. The axioms are also sound as no additional FDs or incorrect FDs can be deduced from F using the rules. All the three axioms are proved for completeness and soundness. Though Armstrong's axioms are complete and sound, computing F + using them directly is a complicated task. Thus, to simplify the task of computing F + from F, additional rules are derived from Armstrong's axioms. These rules are as follows: ? Decomposition

43%	**MATCHING BLOCK 121/319**	SA	248E1130_RDBMS.docx (D165247738)

rule: If A → BC then A → B holds and A → C holds. ? Union rule: If A → B and A → C hold, then A → BC holds. ? Pseudotransitivity rule: If A → B holds and BC → D holds, then AC → D holds.

For example, consider
a relation

79%	**MATCHING BLOCK 122/319**	SA	248E1130_RDBMS.docx (D165247738)

schema R(A, B, C, D, E, G) and the set F of functional dependencies { A → B,

BC → D, D → E, D → G}. Some of the members of F + of the given set F are ? AC → D. Since A → B and BC → D (Pseudotransitivity rule) ? BC → E. Since BC → D and D → E (Transitivity rule) ? D → EG. Since D → E and D → G (Union rule) 6.3.3 Closure of a Set of Attributes In practice, there is little need to compute the closure of set of FDs. Generally, it is required to compute certain subset of the closure (subset consisting of all FDs with a specified set Z of attributes as determinant). However, to compute subset of closure, computing F + is not an efficient process as F + can be large. Moreover, computing F + for a given set F of FDs involves applying Armstrong's axioms repeatedly until they stop generating new FDs. So, some algorithm must be devised that can compute certain subset of the closure of the given set F of FDs without computing F + . For a given relation schema R, a set Z of attributes of R and the set F of functional dependencies that hold on R, the set of all attributes of R that are functionally dependent on Z (known as the closure of Z under F, denoted by Z +) can be determined using the algorithm as shown in Figure 6.5. Z + := Z; //Initialize result to Z while(change in Z +) do // until change in result for each FD X → Y in F do //inner loop begin if X ⊆ Z + //check if left hand side is subset //of closure(result) then Z + := Z + U Y; //include the value on right hand // side in result 152
end Fig. 6.5 Algorithm to Compute Z For example,
consider
a relation

83%	**MATCHING BLOCK 123/319**	SA	248E1130_RDBMS.docx (D165247738)

schema R(A, B, C, D, E, G, H) and the set F of functional dependencies {A → B, BC → DE, AEG → H}. The steps to compute the closure {AC} + of the set of attributes {A, C} under the set F of FDs (using algorithm given in Figure 6.5) are 1. Initialize the result Z + to {A, C}. 2. On the execution of while loop first time, inner loop is repeated three times, once for each FD. 3. On the first iteration of the loop for A → B, since left hand side A is subset of Z + , include B in Z + . Thus, Z + = {A, B, C}. 4. On the second iteration of the loop for BC → DE, include DE in Z + . Now Z + = {A, B, C, D, E}. 5. On the third iteration of the loop for AEG → H, Z + remains unchanged, since AEG is not subset of Z + . 6. On the execution of while loop second time, inner loop is repeated three times again. In all iterations, no new attribute is added in Z + , hence, the algorithm terminates. Thus, {AC} + = {A, B, C, D, E}. Besides, computing subset of closure, the closure algorithm has some other uses which are as follows. ? To determine if a particular FD, say X → Y is in the closure F + of F, without computing the closure F+. This can be done by simply computing X + by using closure algorithm and then checking if Y ⊆ X + . ? To test if a set of attributes A is a superkey of R. This can be done by computing A + , and checking if A + contains all the attributes of R. 6.3.4 Equivalence of Sets of Functional Dependencies Before discussing the equivalence sets of dependencies, one must be familiar with the term cover. A set F 2 of FDs is covered by another set F 1 or F 1 is cover of F 2 if every FD in F 2 can be inferred from F 1 , that is, if every FD in F 2 is also in F 1 + . This means, if any database system enforces FDs in F1, then FDs in F 2 will be enforced automatically. Two sets F 1 and F 2 of FDs are said to be equivalent if F 1 + = F 2 + , that is, every FD in F 1 is implied by F 2 and every FD in F 2 is implied by F 1 . In other words, F 1 is equivalent to F 2 , if both the conditions F1 covers F 2 and F 2 covers F 1 hold. It is clear from the definition, that if F 1 and F 2 are equivalent and the FDs in F 1 are enforced by any database system, then the FDs in F 2 will be enforced automatically and vice versa. 6.3.5 Canonical Cover For every set of FDs, if a simplified set that is equivalent to given set of FDs can be generated, then the effort spent in checking the FDs can be reduced on each update. Since, simplified set will have the same closure as that of

70%	MATCHING BLOCK 124/319	SA	248E1130_RDBMS.docx (D165247738)

given set, any database that satisfies the simplified set of FDs will also satisfy the given set and vice versa.

For example, consider a relation schema R(A, B, X, Y) with a set F of FDs AB → XY and A → Y. Now, whenever an update operation is applied on the relation, the database verifies both the FDs. If we carefully examine the above given FDs, then we find that if we remove the attribute Y from the right-hand side of AB → XY, then it doesn't affect the closure of F, since AB → X and A → Y implies AB → XY (by union rule). Thus, verifying AB → X and A → Y means same as that of verifying AB → XY and A → Y. However, verifying AB → X and A → Y requires less effort than checking the original set of FDs. This attribute X is extraneous in the dependency AB → XY, since the closure of new set is same as the closure of F. Formally, extraneous attribute can be defined as follows. Consider a functional dependency X → Y in a given set F of FDs. ?

52%	MATCHING BLOCK 126/319	SA	248E1130_RDBMS.docx (D165247738)

Attribute A is extraneous in X if A ∊ X, and F logically implies (F−{X → Y}) ∪ {(X−A) → Y}. ? Attribute A is extraneous in Y if A ∊ Y, and the set of FDs (F−{X → Y}) ∪ {X → (Y−A)} logically implies F. 153

Once all such extraneous attributes are removed from the FDs, a new set say F c is obtained. In addition, if the determinant of each FD

87%	MATCHING BLOCK 125/319	W

in F c is unique, that means, there are no two dependencies

like X → Y and X → Z in F c , then F c is called canonical cover for F.

95%	MATCHING BLOCK 127/319	SA	248E1130_RDBMS.docx (D165247738)

A canonical cover for a set of functional dependencies can be computed by using the algorithm given in Figure 6.6. F c := F; while(change in F c) do replace any FD in F c of the form X → A, X → B with X → AB (Use union rule) for(each FD X → Y in F c)do begin if(extraneous attribute is found either in X or in Y) then remove it from X → Y end Fig. 6.6 Algorithm to compute canonical cover for F To illustrate the algorithm given in Figure 6.6, consider a set F of functional dependencies {A → C, AC → D, E → AD, E → H} on relation schema R(A,B,C). To compute the canonical cover for a set F, follow these steps. 1. Initialize F c to F. 2. On first iteration of the loop, replace FDs E → AD and E → H with E → ADH, as both these FDs have same set of attributes in determinant (left side of FD). Now, F c = {A → C, AC → D, E → ADH}. In FD AC → D, C is extraneous. Thus, F c = {A → C, A → D, E → ADH}. 3. On second iteration, replace FDs A → C and A → D with A → CD. Now, F c = {A → CD, E → ADH}. In FD A → CD, no attribute is extraneous. In FD E → ADH, D is extraneous. Thus, F c = {A → CD, E → AH}. 4. On third iteration, the loop terminates, since determinant of each FD is unique and no extraneous attribute is there. Thus, F c (canonical cover) is {A → CD and E → AH}. It is obvious that the canonical cover of F, F c has same closure as F. This implies, if we have determined that F c is satisfied, then we can say F is also satisfied. Moreover, F c is minimal as FDs with same left side have been combined and F c does not contain extraneous attributes. Thus, it is easier to test F c than to test F itself. 6.4 NORMAL FORMS Consider a relation R with a designated primary key and a set F of functional dependencies. To decide whether the relation is in desirable form or it should be decomposed into smaller relations, some guidelines or formal methodology is needed (as stated earlier). To provide such guidelines, several normal forms have been defined (see Figure 6.7). Fig. 6.7

88%	MATCHING BLOCK 128/319	SA	DCAP 011.docx (D142453284)

Normal forms 154 A relation is said to be in a particular normal form if it satisfies certain

specified constraints. Each of the normal forms is stricter than its predecessors.

58%	MATCHING BLOCK 129/319	SA	DBMS Block 2.pdf (D149011992)

The normal forms are used to ensure that various types of anomalies and inconsistencies are removed from the database. Initially, Codd proposed three normal forms namely, first normal form (1NF), second normal form (2NF),

and third normal form (3NF).

There were certain inadequacies in the third normal form, so a revised definition was given by Boyce and Codd. This new definition is known as Boyce-Codd normal form (BCNF) to distinguish it from the old definition of third normal form. This new definition is stricter than the old definition of third normal form as any relation which is in BCNF is also in third normal form; however, reverse may not be true. All these normal forms are based on the functional dependency. Two other normal forms, namely, fourth normal form (4NF) and fifth normal form (5NF) have been proposed which are based on the concepts of multivalued dependency and join dependency, respectively. The process of modifying a relation schema so that it conforms to certain rules (normal forms) is called normalization. Normalization is conducted by evaluating a relation schema to check whether it satisfies particular rules and if not, then decomposing schema into set of smaller relation schemas that do not violate those rules. Normalization can be considered as fundamental to the modeling and design of a relational database, the main purpose of which is to eliminate data redundancy and avoid data update anomalies. Things to Remember E. F. Codd introduced 1NF in 1970, and 2NF and 3NF in 1971. BCNF was proposed by E. F. Codd and Raymond F. Boyce in 1974. Other normal forms upto DKNF were introduced by Ronald Faign. He defined 4NF, 5NF, and DKNF in 1977, 1979, and 1981, respectively. A relation in a lower normal form can always be reduced to a collection of relations in higher normal form. A relation in higher normal form is improvement over its lower normal form. Generally, it is desirable to normalize the relation schema in a database to the highest possible form. However, normalizing a relation schema to higher forms can introduce complexity to the design and application. As it results in more relations and more relations need more join operations while carrying out queries in database system which can decrease the performance. Thus, while normalization, performance should be taken into account that may result in a decision not to normalize beyond third normal form or BCNF normal form. NOTE Redundancy cannot be minimized to zero in any database system. 6.4.1 First Normal Form The first normal form states that every tuple must contain exactly one value for each attribute from the domain of that attribute. That is, multivalued attributes, composite attributes, and their combinations are not allowed in 1NF. Definition:
A relation schema

96%	MATCHING BLOCK 131/319	SA	DCAP 011.docx (D142453284)

R is said to be in first normal form (1NF) if and only if

the domains of all attributes of R contain atomic (or indivisible) values only. Consider the relation schema BOOK_INFO{ISBN, Price, Page_count, P_ID, R_ID, rating} that represents the information regarding books and reviews. The functional dependencies that hold on relation schema BOOK_INFO are {ISBN → Price, Page_count, P_ID} and {ISBN, R_ID → Rating}. The only key of the relation BOOK_INFO is the combination of ISBN and R_ID. For the purpose of explaining normalization, an additional constraint Page_count → Price is introduced. It implies that price of book is determined by the number of pages in the book. The FD diagram for BOOK_INFO is shown in Figure 6.8 and corresponding instance is shown in Figure 6.9. Fig. 6.8 FD diagram for BOOK_INFO 155
Fig. 6.9 Unnormalized relation BOOK_INFO It is clear from Figure 6.9 that tuples contain multiple values for some of the attributes. For example, the tuple for ISBN 001-987-760-9 has two values for the attributes R_ID and Rating. Thus, it is an unnormalized relation. A relation is said to be an unnormalized relation if it contains non-atomic values. This relation can be normalized into 1NF by creating one tuple for each value in multivalued attributes as shown in Figure 6.10. Fig. 6.10 Relation BOOK_INFO in 1NF A good database design always requires that domains of all the attributes of a relation must be atomic, so that the relation schemas are in 1NF. However, depending on the requirement or from the performance point of view, sometimes non- atomic values are useful. For instance, a single attribute Address can be used to hold house number, street, city such as 12, Park street, Atlanta in a relation. Whether the house number, the street name, and city should be separated into distinct attributes depends on how data is to be used. For example, if the address for mailing is needed only, then a single 156
attribute Address can be used. However, if the information like all the publishers in a given street is required, then house number, the street name, and city should constitute their own attributes. NOTE Modern database system supports non-atomic values. Though the BOOK_INFO is in 1NF, it has some shortcomings. The information regarding the same book is repeated a number of times. Further, there are certain anomalies with respect to update operations. For example, the information of a new book cannot be inserted unless some reviewer reviews it. In addition, the change in the price of a book for particular ISBN can lead to inconsistency in the database, if it is not changed in all the tuples where the same ISBN appears. Moreover, the deletion of information of rating for a book having ISBN, say, 003-456-533-8 can cause the loss of detail of the book. This is because there exists only one tuple that contains this particular ISBN. Thus, to resolve these problems, this schema needs further normalization. 6.4.2

97%	MATCHING BLOCK 130/319	SA	Advanced DBMS.pdf (D166063498)
Second Normal Form (2NF) The second normal form is based on the concept of full functional dependency.			

An attribute Y of a relation schema R is said to be fully functional dependent on attribute X (X → Y), if there is no A, where A is proper subset of X such that A → Y. It implies removal of any attribute from X means that the dependency does not hold any more. A dependency X → Y is a partial dependency if there is any attribute A where A is proper subset of X such that A → Y. For example, consider the relation schema BOOK_INFO in which the attributes ISBN and R_ID together form the key. In this schema, the attribute Rating is fully functional dependent on the key {ISBN, R_ID} as Rating is not dependent on any subset of key. That is, neither ISBN → Rating nor R_ID → Rating holds. However, the attributes Price, Page_count, P_ID are partially dependent on key, since they are dependent on ISBN, which is a subset of key {ISBN, R_ID}. Definition:
A relation schema R is said to be

91%	MATCHING BLOCK 132/319	W	
in second normal form (2NF) if every non-key attribute A in R is fully functionally dependent on			

the primary key.
An attribute is non-key or non-

76%	MATCHING BLOCK 134/319	SA	Database System.pdf (D165747767)
prime attribute, if it is not a part of candidate key of R.			

Consider once again, the relation schema BOOK_INFO. This schema is not in 2NF as it involves partial dependencies. This schema can be normalized to 2NF by eliminating all partial dependency between a non-key attribute and the key. This can be achieved by decomposing BOOK_INFO into two relation schemas as follows: BOOK{ISBN,Price, Page_count, P_ID} REVIEW {ISBN,R_ID, Rating} The instances corresponding to the relation schemas BOOK and REVIEW are shown in Figure 6.11. 157

Fig. 6.11 Relations in 2NF The relation schemas BOOK and REVIEW do not have any partial dependencies. Each of the nonkey attributes of relation schemas BOOK and REVIEW is fully functional dependent on key attribute ISBN and (ISBN, R_ID), respectively. Hence, both the schemas are in 2NF. The FDs corresponding to these relation schemas are shown in Figure 6.12. Fig. 6.12 FDs in 2NF Now, all the problems of BOOK_INFO have been overcome by normalizing it into 2NF; however, the decomposed relation schema BOOK which is in 2NF still causes problems over update operation. For example, the Price for Page_count 900 cannot be inserted unless there exists a book having 900 pages. Similarly, the change in the Price for particular Page_count can lead to inconsistency in the database if all the tuples where the same Page_count appears are not changed. Moreover, the deletion of the Price for particular Page_count say, 200 can cause the loss of information of the book having ISBN 001- 354-921-1. This is due to the fact that there exists only one tuple that contains the information of book having this particular ISBN. These problems can be resolved by normalizing this relation schema further. Since, relation schema REVIEW do not have such type of anomalies, there is no need to normalize it further. Learn More While normalizing the relation schemas, sometimes, it is required to introduce artificial key (also known as surrogate key) in some decomposed relations. It is used when a primary key is not available or inapplicable. Testing a relation schema for 2NF involves testing FDs where determinant is part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. 158
6.4.3 Third

84%	MATCHING BLOCK 133/319	SA	Advanced DBMS.pdf (D166063498)

Normal Form (3NF) The third normal form is based on the concept of

transitive dependency.
An attribute Y of a relation schema R is said to be transitively dependent on attribute X (X → Y), if there is set of attributes A

82%	MATCHING BLOCK 135/319	SA	BOOK SIZE.docx (D50092775)

that is neither a candidate key nor a subset of any key of R and both X →

A and A → Y hold. For example, the dependency ISBN → Price is transitive through Page_count in relation schema BOOK. This is because both the dependencies ISBN → Page_count and Page_count → Price hold and Page_count is neither a candidate key nor a subset of candidate key of BOOK.
Definition:

86%	MATCHING BLOCK 136/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

A relation schema R is in third normal form (3NF) with a set F of functional dependencies if, for

every
FD X → Y in F,
where X be any subset of attributes of R and Y be any single attribute of

48%	MATCHING BLOCK 138/319	SA	248E1130_RDBMS.docx (D165247738)

R, at least one of the following statements hold ? X → Y is a trivial FD, that is, Y ⊆ X. ? X is a superkey for R. ?

Y is contained in a candidate key for R.
It can also be stated as,
A relation schema

84%	MATCHING BLOCK 139/319	SA	Unit-4 (Part-II).docx (D76435895)

R is in third normal form (3NF) if and only if it satisfies 2NF and every nonkey attribute is non-transitively dependent on the primary key.

For example, the schema BOOK can be decomposed into two relation schemas say, BOOK_PAGE and PAGE_PRICE to eliminate transitive dependency between the attributes ISBN and Price. Hence, the schemas BOOK_PAGE and PAGE_PRICE are in 3NF. The instances of these schemas are shown in Figure 6.13, and Figure 6.14 shows FD diagrams of these schemas. Note that this definition of 3NF can be used to handle relations in which primary key is the only candidate key. Thus, to handle the relations having more than one candidate key, this old definition of 3NF was replaced by new definition by Boyce and Codd normal form which is also known as BCNF. Fig. 6.13 Relations in 3NF Fig. 6.14 FD diagram for BOOK_PAGE and PAGE_PRICE 6.4.4 Boyce-Codd Normal Form (BCNF) Boyce-Codd normal form was proposed to deal with the relations having two or more candidate keys that are composite and overlap. Two candidate keys overlap if they contain two or more attributes and have at least one attribute in common. 159
Definition:

A relation schema R is in Boyce-Codd normal form (BCNF) if, for every FD X → A in F, where X is

the subset of the attributes of R, and A is an attribute of R, one of the following statements holds ? X → Y is a trivial FD, that is, Y ⊆ X. ? X is a superkey. In simple terms, it can be stated as A relation schema R is in BCNF if and only if every non-trivial FD has a candidate key as its determinant. For example, consider a relation schema BOOK(ISBN, Book_title, Price, Page_count) with two candidate keys, ISBN and Book_title. Further, assume that the functional dependency Page_count → Price no longer holds. The FD diagram for this relation schema is shown in Figure 6.15. As shown in Figure 6.15, there are two determinants ISBN and Book_title and both of them are candidate keys. Thus, this relation schema is in BCNF. BCNF is the simpler form of 3NF as it makes explicit reference to neither the first and second normal forms, nor to the concept of transitive dependence. Furthermore, it is stricter than 3NF as every relation which is in BCNF is also in 3NF, but vice versa is not necessarily true. A 3NF relation will not be in BCNF if there are multiple candidate keys in the relation such that ? the candidate keys in the relation are composite keys, and ? the keys are not disjoint, that is, candidate keys overlap. To illustrate this point, consider a relation schema BOOK_RATING(ISBN, Book_title, R_ID, Rating). The candidate keys are (ISBN, R_ID) and (Book_title, R_ID). This relation schema is not in BCNF since both the candidate keys are composite as well as overlapping. However, it is in 3NF. Fig. 6.15 FD diagrams for BOOK and with Book_title as candidate key Fig. 6.16 Instance of relation BOOK_RATING 160
From the instance corresponding to this relation schema as shown in Figure 6.16, it is clear that it suffers from the problem of repetition of information. Thus, changing the title of book can cause logical inconsistency if it is not changed in all the tuples in which it appears. This problem can be resolved by decomposing this relation schema into two relation schemas as shown here. BOOK_TITLE_INFO(ISBN, Book_title) and REVIEW(R_ID, ISBN, Rating) Or BOOK_TITLE_INFO(ISBN, Book_title) and REVIEW(R_ID, Book_title, Rating) Now, all these relation schemas are in BCNF. Note that BCNF is the most desirable normal form as it ensures the elimination of all redundancy that can be detected using functional dependencies. If there is only one determinant upon which other attributes depend and it is a candidate key, 3NF and BCNF are identical. 6.5 INSUFFICIENCY OF NORMAL FORMS From the discussion so far, it can be concluded that generally normalization start from a single universal relation which includes all attributes of a database. The relation is decomposed repeatedly until a BCNF or 3NF is achieved. However, this condition is not sufficient on its own to guarantee a good relational database schema design. In addition to this condition, there are two additional properties that must hold on decomposition to qualify it as a good design. These are lossless-join property and dependency preservation property. 6.5.1 Lossless-Join Property The notion of the lossless-join property is crucial as it ensures that the original relation must be recovered from the smaller relations that result after decomposition. That is, it ascertains that no information is lost after decomposition. Definition: Let R be a relation schema with a given set F of functional dependencies. Then decomposition D = {R 1 , R 2 , ..., R n } on R is said to be lossless (lossless–join), if for every legal relation (that is, relation that satisfies the specified functional dependencies and other constraints), following statement holds. Π R1 (R) Π R2 (R) ... Π Rn (R)= R In other words, if the original relation can be reconstructed by combining the projections of a relation using natural join (or the process of decomposition is reversible), decomposition is lossless decomposition, otherwise, it is a lossy (lossy–join) decomposition. For example, consider again the relation schema BOOK(ISBN, Price, Page_count) with the functional dependencies ISBN → Price and Page_count → Price. Other attributes are ignored to simplify the relation schema. An instance of this relation schema is shown in Figure 6.17. The relation schema BOOK(ISBN, Price, Page_count) can be decomposed in two different ways, say, decomposition A and decomposition B (see Figure 6.17). The schemas corresponding to decomposition A and decomposition B are given here. Decomposition A BOOK_PRICE(ISBN, Price) and BOOK_PAGES(ISBN, Page_count) Decomposition B BOOK_PRICE(ISBN, Price) and PRICE_PAGE(Price, Page_count) 161

Fig. 6.17 Instance of relation BOOK and two corresponding decompositions In Decomposition A (see Figure 6.17), the price and the number of pages for all the books can be determined easily. Since, no information is lost in this decomposition, decomposition A is lossless. In Decomposition B (see Figure 6.17), it can be determined that price of the books having ISBN 001-987-760-9 and 002- 678-980-4 is 25. However, the number of pages in these books cannot be determined as there are two values for page_count in relation PRICE_PAGE corresponding to the Price 25. Thus, this decomposition is lossy. If an decomposition does not have the lossless-join property, then additional spurious tuples (tuples that are not in original relation) may be obtained on applying join operation on the relations resulted after decomposition. These spurious tuples represent erroneous data. For example, consider the relations BOOK_PRICE and PRICE_PAGE as shown in decomposition B in Figure 6.17. If these relations are joined, a new relation having additional spurious tuples is obtained as shown in Figure 6.18. Fig. 6.18 Relation with spurious tuples However, if the relations as shown in decomposition A in Figure 6.17 are joined, no spurious tuples will be generated. Thus, if lossless-join property holds on decomposition, it is ensured

that no spurious tuples are obtained when a natural join operation is applied to the relations

resulted after decomposition. So, this property is also known as non-additive join property. A particular decomposition can be tested for having lossless property on the basis of functional dependency by applying the property as given here. Lossless-join test for binary decompositions: Let R be a relation schema with F a set of FDs over R. The

decomposition of R into two relations R 1 and R 2 form a lossless decomposition if at least one of the following dependencies is in F + . R 1 ∩ R 2 → R 1 – R 2 R 1 ∩ R 2 →

R 1 – R 2
This property is limited to binary decompositions (decomposition of a relation into two relations) only. In addition,

it is a necessary condition only if all constraints are functional dependencies. 162

NOTE The test for lossless decomposition in case of a decomposition of a relation into multiple relations is beyond this discussion. 6.5.2 Dependency Preservation Property Another important property that is desired while decomposition is dependency preservation, that is, no FD of the original relation is lost. Consider a relation schema R with set F of FDs {X → Y, Y → Z}. One other dependency that is implied from F is X→ Z. Let R is decomposed into two relation schemas, R 1 {X,Y} with FD X → Y and R 2 {Y,Z} with FD Y → Z. Here, X → Z is enforced automatically since it is implied from FDs of R 1 and R 2 . Hence, this decomposition is dependency preserving. Now, suppose that R is decomposed into two relation schemas, R 1 {X,Y} with FD X → Y and R 2 {X,Z} with FD X → Z. Here, FD Y → Z is not implied from R 1 and R 2 . To enforce this FD, join operation is to be applied whenever a new tuple is to be inserted. We know that computing join is a costly process in terms of CPU usage and response time. Dependency preservation property ensures that each FD represented by original relation is enforced by examining any single relation resulted from decomposition or can be inferred from the FDs in some decomposed relation. Thus, it prevents computation of join. To understand this property, let us first understand the term projection. Given a relation schema R with set F of FDs and R 1 , R 2 ,..., R n be the

decomposition of R, the projection of F to R i is the set F i of all FDs in F + (not just F) that include only attributes of

R i . For example, once again consider the decomposition of R 1 {X,Y} and R 2 {X,Z} of R. The projection of F to R 1 , denoted by F R1 is X → Z, since X → Z in F + , even though it is not in F. This FD can be tested by examining relation R 2 only. Similarly, FDs in each projection can be tested by analyzing only one relation in the decomposed set. However, testing each projection is not sufficient to determine whether the decomposition is dependency preserving. Since, it is not always necessary that all the dependencies in original relation appear directly in an individual decomposed relation. That is, there may be FDs in F + that are not the projections. Thus, we also need to examine the FDs that are inferred from the projections. Formally, the decomposition of relation schema R with a given set F

of FDs into relations R 1 , R 2 , ..., R n is dependency preserving if F′ + = F + , where F′= F

R1 ∪ F R2 ∪... ∪ F Rn . Lossless-join property is indispensable requirement of decomposition that must be achieved at any cost, whereas the dependency preservation property, though desirable, can be sacrificed sometimes. 6.6 COMPARISON OF BCNF AND 3NF From the discussion so far, it can be concluded that the goal of a database design with functional dependencies is to obtain schemas that are in BCNF, are lossless and preserves the original set of FDs. A relation can always be decomposed into 3NF in such a way that the decomposition is lossless and dependency preserving. However, there are some disadvantages to 3NF. There is a possibility that null values are used to represent some of the meaningful relationships among data items. In addition, there is always a scope of repetition of information. On the other hand, it is always not possible to obtain a BCNF design without sacrificing dependency preservation. For example, consider a relation schema CITY_ZIP(City, Street, Zip) with FDs {City, Street} → Zip and Zip → City is decomposed into two relation schemas R 1 (Street, Zip) and R 2 (City, Zip). It is observed that both these schemas are in BCNF. In R 1 , there is only trivial dependency and in R 2 , the dependency Zip → City holds. It can be seen clearly that the FD {City, Street} → Zip is not implied from decomposed relation schemas. Thus, this decomposition does not preserve dependency, that is, (F 1 ∪ F 2) + ≠ F + , where F 1 and F 2 are the restrictions of R 1 and R 2 , respectively. Clearly, it is not always possible to achieve the goal of database design. In that case, we have to be satisfied with relation schemas in 3NF rather than BCNF. This is because, if dependency preservation is not checked efficiently, either system performance can be reduced or it may risk the integrity of the data. Thus, maintaining the integrity of data at the cost of the limited amount of redundant data is a better option. 6.7 HIGHER NORMAL FORMS As stated earlier, BCNF is the desirable normal form; however, some relation schemas even though they are in BCNF still suffer from the problem of repetition of information. For example, consider an alternative design for Online Book database 163
having a relation schema BOOK_AUTHOR_DETAIL(ISBN, A_ID, Phone). An instance of this relation schema is shown in Figure 6.19(a). In this relation, there are two multivalued attributes, A_ID and Phone that represents the fact that a book can be written by multiple authors and an author can have multiple phone numbers. That is, for a given ISBN, there can be multiple values in attribute A_ID and for a given A_ID, there can be multiple values for attribute Phone. To make the relation consistent, every value of one of the attributes is to be repeated with every value of the other attribute [see Figure 6.19(b)]. Fig. 6.19 Relation BOOK_AUTHOR_DETAIL From Figure 6.19(b), it is clear that relation BOOK_AUTHOR_DETAIL satisfies only trivial dependencies. In addition, it is all key relation (a relation in which key is the combination of all the attributes taken together) and any all key relation must necessarily be in BCNF. Thus, this relation is in BCNF. However, it involves lot of redundancies which leads to certain update anomalies. Thus, it is always not possible to decompose a relation on the basis of functional dependencies to avoid redundancy. NOTE BCNF can eliminate the redundancies that are based on the functional dependencies. Since, normal forms based on functional dependencies are not sufficient to deal such type of situations, other type of dependencies and normal forms have been defined. 6.7.1 Multivalued Dependencies Multivalued dependencies (MVDs) are the generalization of functional dependencies. Functional dependencies prevent the existence of certain tuples in a relation. For example, if a FD X → Y holds in R, then any r(R) cannot have two tuples having same X value but different Y values. On the contrary, multivalued dependencies do not rule out the presence of those tuples, rather, they require presence of other tuples of a certain form in the relation. Multivalued dependencies arise when a relation having a non-atomic attribute is converted to a normalized form. For each X value in such a relation, there exists a well-defined set of Y values associated with it. This association between the X and Y values does not depend on the values of the other attributes in the relation. 164

To understand the notion of multivalued dependencies more clearly, consider the relation BOOK_AUTHOR_DETAIL as shown in Figure 6.19(a). In this relation, although the ISBN does not have a unique A_ID (the FD ISBN → A_ID does not hold), each ISBN has a well defined set of corresponding A_ID. Similarly for a particular A_ID, a well defined set of Phone exists. In addition, book is independent of the phone numbers of authors, due to which there is lot of redundancy in this relation. Such situation which cannot be expressed in terms of FD is dealt by specifying a new form of constraint called multivalued dependency or MVD. Definition: In a relation schema R, an attribute Y is said to be multi-dependent on attribute X (X →→ Y) if and only if for a particular value of X, the set of values of Y is completely determined by the value of X alone and is independent of the values of Z where, X, Y, and Z are the subsets of the attributes of R. X →→ Y is read as Y is multi-dependent on X or X multi-determines Y. From the definition of MVD, it is clear that if X → Y, then X →→ Y. In other words, every functional dependency is a multivalued dependency. However, the converse is not true. The multivalued dependencies that hold in the relation BOOK_AUTHOR_DETAIL, [see Figure 6.19(b)] can be represented as ISBN →→ A_ID A_ID →→ Phone If the MVD X →→ Y

80%	MATCHING BLOCK 145/319	SA	248E1130_RDBMS.docx (D165247738)

is satisfied by all relations on schema R, then X →→ Y is a trivial dependency on schema R,

that is, X →→ Y is trivial if Y ⊆ X or Y ∪ X = R. NOTE Generally, the relations containing non-trivial MVDs are all key relations. Like functional dependencies (FDs), inference rules for multivalued dependencies (MVDs) have been formulated which are complete as well as sound. These set of rules consist of three Armstrong axioms (discussed earlier) and five additional rules. Three of these rules involve only MVDs which are ? Complementation rule for MVDs: If X →→ Y, then X →→ R–XY. ? Augmentation rule for MVDs: If X →→ Y and W ⊇ Z, then WX →→ YZ. ? Transitive rule for MVDs: If X →→ Y and Y →→ Z, then X →→ (Z–Y). The remaining two inference rules relate FDs and MVDs. These rules are ? Replication rule for FD to MVD: If X → Y, then X →→ Y. ? Coalescence rule for FD to MVD: If X →→ Y and there exists W such that W ∩ Y is empty, W → Z and Y ⊇ Z, then X → Z. Multivalued dependency is a result of first normal form which disallows an attribute to have multiple values. If a relation has two or more multivalued independent attributes, and is independent of one another, it is specified using multivalued dependency. 6.7.2 Fourth Normal Form Fourth normal form is based on the concept of multivalued dependency. It is required when undesirable multivalued dependencies occur in a relation. This normal form is more restrictive than BCNF. That is, any 4NF is necessarily in BCNF but converse is not true. Definition:

91%	MATCHING BLOCK 146/319	SA	248E1130_RDBMS.docx (D165247738)

A relation schema R is in fourth normal form (4NF) with respect to a set F of functional and multivalued dependencies if, for

every
non-trivial multivalued dependency
X →→ Y in F + ,
where X ⊆ R and Y ⊆ R, X is a superkey of R. For example, consider, once again relation BOOK_AUTHOR_DETAIL. As stated earlier, the problem is due to the fact that book is independent of the phone number of authors. This problem can be resolved by eliminating this undesirable dependency, that is, by decomposing this relation into two relations BOOK_AUTHOR and AUTHOR_PHONE as shown in Figure 6.20. 165
Fig. 6.20 Relations in 4NF Since, the relations BOOK_AUTHOR and AUTHOR_PHONE do not involve any MVD, they are in 4NF. Hence, this design is an improvement over the original design. Further, relation BOOK_AUTHOR_DETAIL can be recovered by joining BOOK_AUTHOR and AUTHOR_PHONE together. Note that whenever a relation schema R is decomposed into relation schemas

89%	MATCHING BLOCK 147/319	SA	BOOK SIZE.docx (D50092775)

R 1 = (X ∪ Y) and R2 = (R–Y) based on MVD X →→ Y that holds in R, the decomposition has the lossless-join property. This condition is necessary and sufficient for decomposing a schema into two schemas that

have lossless-join property. Formally, this condition can be stated as: Property for lossless-join decomposition into 4NF:

95%	MATCHING BLOCK 149/319	SA	BOOK SIZE.docx (D50092775)

The relation schemas R 1 and R 2 form lossless-join decomposition of R

87%	MATCHING BLOCK 148/319	W	

with respect to a set F of functional and multivalued dependencies if

and only if (R 1 ∩ R 2) →→ (R 1 - R 2) or (R 1 ∩ R 2) →→ (R 1 - R 2) 6.7.3 Join Dependencies The property (lossless-join test for binary decompositions) that is discussed earlier, gives the condition for a relation schema to be decomposed in a lossless way by exactly two of its projections. However, in some cases, there exist relations that cannot be lossless decomposed into two projections but can be lossless decomposed into more than two projections. Moreover, it may be in 4NF that means there may neither any functional dependency in R that violates any normal form upto BCNF nor any non-trivial dependency that violates 4NF. However, the relation may have another type of dependency called join dependency which is the most general form of dependency possible. Consider a relation AUTHOR_BOOK_PUBLISHER (see Figure 6.21) which is all key relation (all the attributes form key). Since, in this relation, no non-trivial FDs or MVDs are involved, it is in 4NF. It should be noted that if it is decomposed exactly into any two relations and joined back, a spurious tuple is created (see Figure 6.21); however, if it is decomposed into three relations and joined back as shown in Figure 6.21, original relation is obtained. It is because it specifies another constraint according to which, whenever an author a writes book b, and a publisher p publishes book b, and the author a writes at least one book for publisher p, then author a will also be writing book b for publisher p. For example, If a tuple having values &gt;A001, C++&lt; exists in AUTHOR_BOOK and a tuple having values &gt;C++, P001&lt; exists in BOOK_PUBLISHER and a tuple having values &gt;P001, A001&lt; exists in PUBLISHER_AUTHOR then a tuple having values &gt;A001, C++, P001&lt; exists in AUTHOR_BOOK_PUBLISHER 166
Note that the converse of this statement is true, that is, if &gt;A001, C++, P001&lt; appears in AUTHOR_BOOK_PUBLISHER, then &gt;A001, C++&lt; appears in relation AUTHOR_BOOK and so on. Further, &gt;A001, C++&lt; appears in AUTHOR_BOOK if and only if &gt;A001, C++, P002&lt; appears in AUTHOR_BOOK_PUBLISHER for some P002 and similarly, for &gt;C++, P001&lt; and &gt;P001, A001&lt;. This statement can be rewritten as a constraint on AUTHOR_BOOK_PUBLISHER as "If tuples having values &gt;A001, C++, p002&lt; &gt;A002, C++, P001&lt; and &gt;A001, Unix, P001 &lt; exists in AUTHOR_BOOK_PUBLISHER then tuple having values &gt;A001, C++, P001&lt; also exists in AUTHOR_BOOK_PUBLISHER". Since, this constraint is satisfied if and only the relation is joining of certain projections, this constraint is referred as join dependency (JD). Fig. 6.21 Non-loss decomposition of a relation into three projections Definition:

76%	MATCHING BLOCK 150/319	SA	MCSDSC-2.2 Relational database Management syst ... (D151484310)

Let R be a relation schema and R 1 , R 2 , ..., R n be the decomposition of R, R is said to satisfy

the join dependency *(R 1 , R 2 , ..., R n) (read as "star R 1 , R 2 , ..., R n "), if and only if Π R1 (R) Π R2 (R) ... Π Rn (R) = R In other words, relation schema R satisfies the JD *(R 1 , R 2 , ..., R n) if and only if every legal instance r(R) is equal to join of its projections on R 1 , R 2 , ..., R n . 167
From this definition, it is clear that MVD is a special case of a JD, where n = 2 (or join dependency is generalization of MVD). A join dependency *(R 1 , R 2 , ..., R n) specified on a relation schema R is a trivial JD, if at least one relation schema R i in JD *(R 1 , R 2 , ..., R n) is the set of all attributes of R (that is, one of the relation schemas R i in JD *(R 1 , R 2 , ..., R n) is equal to R). Such a dependency is called trivial because it has the lossless-join property for any instance of R and hence, does not specify any constraint on R. 6.7.4 Fifth Normal Form Fifth Normal Form also called project–join normal form (PJ/NF) is based on the concept of join dependencies. Definition:

85%	MATCHING BLOCK 151/319	SA	BOOK SIZE.docx (D50092775)

A relation schema R is in fifth normal form (5NF) with respect to a set F of functional, multivalued, and join dependencies if, for every non-trivial join dependency *(R 1 , R 2 , ..., R n) in F + , every R i is a superkey of R.

For example, consider once again the AUTHOR_BOOK_PUBLISHER relation (see Figure 6.21) which satisfies join dependency that is not implied by its sole superkey (that key being the combination of all its attributes). Thus, it is not in 5NF. In addition, this relation suffers from a number of update anomalies. These anomalies can be eliminated if it is decomposed into three relations AUTHOR_BOOK, BOOK_PUBLISHER, and PUBLISHER_AUTHOR. Each of these relations is in 5NF since they do not involve any join dependency. Further, if join operation is applied on any two of these relations, spurious tuples are generated; however, it is not so on applying join operation on all three relations. Note that any relation in 5NF is automatically in 4NF. Learn More There exists another normal form called domain-key normal form (DK/NF), which is based on domain, key, and general constraints. It is considered as the highest form of normalization. However, it is not always possible to generate schemas in this form. Thus, it is rarely used. Unlike FDs and MVDs, it is difficult to detect join dependencies as there is no set of complete and sound inference rules for this constraint. Thus, 5NF is rarely achievable. 6.8 DENORMALIZATION From the discussion so far, it is clear that when the process of normalization is applied to a relation, the redundancy is reduced. However, it results in more relations which increase the CPU overhead. Hence, it fails to provide adequate response time, that is, makes the database system slow which often counteracts the redundant data. Thus, sometimes, in order to speed up database access, the relations are required to be taken from higher normal form to lower normal form. This process of taking a schema from higher normal form to lower normal form is known as denormalization. It is mainly used to improve system performance to support time-critical operations. There are several other factors which require denormalizing relations are given here. ? Many critical queries that require data to be retrieved from more than one relation. ? Many computations need to be applied to one or many attributes before answering a query. ? The values in multivalued attributes need to be processed in a group instead of individually. ? Relations need to be accessed in different ways by different users during same duration. Like normalization, there is no formal guidance that outlines the approach to denormalizing a database. Once the process of denormalization is started, it is not clear where should it be stopped. Further, the relations that are denormalized suffer from a number of other problems like redundancy and update problems. It is obvious as the relations, which are to be dealt are once again the relations that are less than fully normalized. Moreover, denormalization makes certain queries harder to express. Thus, the main aim of denormalization process is to balance the need for good system response time and need to maintain data while avoiding the various problems associated with denormalized relations. 168 SUMMARY 1. Designing of relation schemas is based on the conceptual design. The relation schemas can be generated directly using conceptual data model such as ER or enhanced ER (EER) or some other conceptual data model. The goodness of the resulting set of schemas depends on the designing of the data model. 2. A relational database is considered as good if it has minimum redundancy and null values. 3. While designing relational database schema, the problems that are confronted due to redundancy and null values can be resolved by decomposition. 4. In decomposition, a relation is replaced with a collection of smaller relations with specific relationship between them. 5. During decomposition, it must be ensured

60%	MATCHING BLOCK 152/319	SA	BOOK SIZE.docx (D50092775)

that each attribute in R must appear in at least one relation schema R i so that no attributes are lost. This is called attribute preservation property of decomposition. 6.

Functional dependencies are one of the most important concepts in relational schema design. They are the special forms of integrity constraints that generalize the concept of keys. A given set of functional dependencies is used in designing a relational database in which most of the undesirable properties are not present. 7. For a given set F of functional dependencies, some other dependencies also hold. That is, some functional dependencies are logically implied by F.

84%	MATCHING BLOCK 153/319	SA	MCSDSC-2.2 Relational database Management syst ... (D151484310)

The set of all FDs that are implied by a given set of FDs is called the closure of F, denoted as F + . 8.

Armstrong proposed a set of inference rules or axioms (also known as Armstrong's axioms) to find logically implied functional dependencies. The axioms are complete and sound. 9. Two sets F 1 and F 2 of FDs are said to be equivalent if F 1 + = F 2 + , that is, every FD in F 1 is implied by F 2 and every FD in F 2 is implied by F 1 . 10. For every set of FDs, canonical cover (a simplified set) that is equivalent to given set of FDs can be generated. 11. To determine the quality or goodness of a database, some formal measures or guidelines are required. Normalization theory provides the guidelines to assess the quality of a database in designing process. The normalization is the application of set of simple rules called normal forms to the relation schema. 12.

86%	MATCHING BLOCK 154/319	SA	DCAP 011.docx (D142453284)
A relation is said to be in a particular normal form if it satisfies certain			

specified constraints. 13. A relation is said to be an unnormalized relation if it contains non-atomic values. 14. Four normal forms, namely,

100%	MATCHING BLOCK 155/319	SA	DBMS.docx (D142535242)
first normal form (1NF), second normal form (2NF), third normal form (3NF), and Boyce-Codd normal form (BCNF)			

are based
on the functional dependency. 15.

100%	MATCHING BLOCK 157/319	SA	DBMS Block 2.pdf (D149011992)
A relation is said to be in first normal form (1NF) if			

the
domains of all its attributes
contain atomic (or indivisible)
values only. 16.

96%	MATCHING BLOCK 156/319	SA	Advanced DBMS.pdf (D166063498)
The second normal form is based on the concept of full functional dependency. 17.			

The

95%	MATCHING BLOCK 158/319	SA	DBMS.docx (D68067493)
third normal form is based on the concept of transitive dependency. 18.			

Boyce-Codd normal form was proposed to deal with the relations having two or more candidate keys that are composite and overlap. 19. There are two additional properties that must hold on decomposition to qualify a relational schema as a good design. These are lossless-join property and dependency preservation property. 20. Lossless-join property ensures that the original relation must be recovered from the smaller relations that result after decomposition. 21. Dependency preservation property ensures that each FD represented by original relation is enforced by examining any single relation resulted from decomposition or can be inferred from the FDs in some decomposed relation. 22. Lossless-join property is indispensable requirement of decomposition that must be achieved at any cost, whereas the dependency preservation property, though desirable, can be sacrificed sometimes. 23. Normal forms based on functional dependencies are always not sufficient to avoid redundancy. So, other types of dependencies and normal forms have been defined. 24. Multivalued dependencies (MVDs) are the generalization of functional dependencies. 25. Fourth normal form is based on the concept of multivalued dependency. It is required when undesirable multivalued dependencies occur in a relation. 169
26. A relation may have another type of dependency called join dependency which is the most general form of dependency possible. 27. Fifth normal form (5NF), also called project–join normal form (PJ/NF) is based on the concept of join dependencies. 28. The process of normalization is applied to a relation to reduce redundancy. However, it results in more relations which increase the CPU overhead. 29. In order to speed up database access, the relations are required to be taken from higher normal form to lower normal form. This process of taking a schema from higher normal form to lower normal form is known as denormalization. 170

CHAPTER 7 DATA STORAGE AND INDEXING After reading this chapter, the reader will understand: ? Different types of storage devices including primary storage, secondary storage, and tertiary storage ? Important characteristics of storage media ? Magnetic disk and its organization ? Seek time, rotational delay, and data transfer time ? Redundant array of independent disk, that is, RAID ? Performance and reliability improvement of disk with RAID ? All the RAID levels from level 0 to 6 ? New storage systems ? The concept of files and pages ? Responsibilities of buffer manager ? Management of buffer space ? Page-replacement policies ? Fixed-length and variable-length records ? Spanned and unspanned organization of records in disk blocks ? Various file organizations including heap file organization, sequential file organization, and hash file organization ? Hashing (both static and dynamic), hash function, and hash table ? Collision and collision resolution ? The concept of bucket ? Primary index, clustering index, and secondary index ? Multilevel indexes including B-tree and B + -tree ? Indexes on multiple keys In the previous chapters, we have viewed the database as a collection of tables (relational model) at the conceptual or logical level. The database users are largely concerned with the logical level of the database as they have hardly to do anything with the physical details of the implementation of the database system. The database is actually stored on a storage medium in the form of files, which is a collection of records consisting of related data items such as name, address, phone, email-id, and so on. In this chapter, we start our discussion with the characteristics of the underlying storage medium. There are various media available to store data; however, database is typically stored on the magnetic disk because of its higher capacity and persistent storage. This chapter also discusses a number of methods used to organize files such as heap, sequential, index sequential, direct, etc., and the issues involved in the choice of a method. Further, the goal of a database system is to simplify and facilitate efficient access to data. One such technique is indexing, which is also discussed in this chapter. 7.1 HIERARCHY OF STORAGE DEVICES In a computer system, several types of storage media such as cache memory, main memory, magnetic disk, etc. exist. On the basis of their characteristics such as cost per unit of data and speed with which data can be accessed, they can be arranged in a hierarchical order (see Figure 7.1). In addition to the speed and cost of the various storage media, another issue is volatility. Volatile storage loses its contents when power supply is switched off, whereas non-volatile storage does not. 171
Fig. 7.1 Storage device hierarchy The cost per unit of data as well as the speed of accessing data decreases while moving down in the memory hierarchy. The storage media at the top such as cache and main memory is the highest speed memory and is referred to as primary storage. The storage media in the next level consists of slower devices, such as magnetic disk, and is referred to as secondary storage. The storage media in the lowest level is the slowest storage devices, such as optical disk and magnetic tape, and are referred to as tertiary storage. Let us discuss these storage devices in detail. Primary Storage Primary storage generally offers limited storage capacity due to its high cost but provides very fast access to data. Primary storage is usually made up of semiconductor chips. Semiconductor memory may be volatile or non-volatile in nature. The two basic forms of semiconductor memory are static RAM or SRAM (used for cache memory) and dynamic RAM or DRAM (used for main memory). ? Cache memory: Cache memory is a static RAM and is very small in size. The cache memory is the fastest; however, most expensive among all the storage media available. It is directly accessed by the CPU and is basically used to speed up the execution. However, major limitation of cache is high cost and its volatility. ? Main memory: Main memory is a dynamic RAM (DRAM) and is used to keep the data that is currently being accessed by the CPU. Its storage capacity typically ranges from 64 MB to 512 MB or sometimes it is extended up to hundreds of gigabytes. It offers fast access to data as compared to other storage media except static RAM. However, it is expensive and offers limited storage capacity. Thus, entire database cannot be placed in main memory at once. Moreover, it is volatile in nature. Another form of semiconductor memory is flash memory. Flash memory uses an electrical erasing technology (like EEPROM). Reading data from flash memory is as fast as reading it from main memory. However, it requires an entire block to be erased and written at once. This makes the writing process little complicated. Flash memory is non-volatile in nature and the data survives system crash and power failure. This type of memory is commonly used for storing data in various small appliances such as digital cameras, MP3 players, USB (Universal Serial Bus) storage accessories, etc. Secondary Storage Secondary storage devices usually provide bulk of storage capacity for storing data but are slower than primary storage devices. Secondary storage is non-volatile in nature, that is, the data is permanently stored and survives power failure and system crashes. Data on the secondary storage are not directly accessed by the CPU. Instead, data to be accessed must 172

move to main memory so that it can be accessed. Magnetic disk (generally called disk) is the primary form of secondary storage that enables storage of enormous amount of data. It is used to hold on-line data for a long term. Generally, the entire database is placed on magnetic disk. Magnetic disk is discussed in detail in Section 7.1.1. Tertiary Storage Removable media, such as optical discs and magnetic tapes, are considered as tertiary storage. Larger capacity and least cost are major advantage of tertiary storage. Data access speed; however, is much slower than primary storage. Since these devices can be removed from the drive, they are also termed as off-line storage. ? Optical disc: Optical disc comes in various sizes and capacities. A compact disc—read only (CD-ROM) with a capacity of 600–700 MB of data having 12 cm diameter is the most popular means of optical disc. Another most common type of optical disc with high-storage capacity is the DVD-ROM, which stands for digital video disc—read only memory. It can store up to 8.5 GB of data per side and DVD allows for double-sided discs. As the name suggests, both the CD-ROM and DVD-ROM come pre-recorded with data, which cannot be altered. However, both use laser beams for performing read operations. ? Tape storage: Tape storage is cheap and reliable storage medium for organizing archives and taking backup. However, tape storage is not suitable for data files that need to be revised or updated often because it stores data in a sequential manner and offers only sequential-access to the stored data. Tape now has a limited role because disk has proved to be a superior storage medium. Furthermore, disk data allows direct-access to the stored data. Table 7.1 lists some of the important characteristics of various storage media available in computer system. Table 7.1 Important characteristics of storage media 7.1.1 Magnetic Disks A magnetic disk is the most commonly used secondary storage medium. It offers high storage capacity and reliability. It can easily accommodate entire database at once. However, if the data stored on the disk needs to be accessed by CPU it is first moved to the main memory and then the required operation is performed. Once the operation is performed, the modified data must be copied back to the disk for consistency of the database. The system is responsible for transferring the data between disk and the main memory as and when required. Data on the disk survives power failures and system crash. There is a chance that disk may sometimes fail itself and destroy the data; however, such failures occur rarely. Data is represented as magnetized spots on a disk. A magnetized spot represents a 1 (bit) and the absence of a magnetized spot represents a 0 (bit). To read the data, the magnetized data on the disk is converted into electrical impulses, which is transferred to the processor. Writing data onto the disk is accomplished by converting the electrical impulses from the processor into magnetic spots on the disk. The data in a magnetic disk can be erased and reused virtually infinitely. The disk is designed to reside in a protective case or cartridge to shield it from the dust and other external interference. Organization of Magnetic Disks A magnetic disk consists of plate / platter, which is made up of metal or glass material, and its surface is covered with magnetic material to store data on its surface. If the data can be stored on only one side of the platter, the disk is single- 173
sided disk, and if both sides are used to hold the data, the disk is double-sided disk. When the disk is in use, the spindle motor rotates the platters at a constant high speed. Usually the speed at which they rotate is 60, 90, or 120 revolutions per second. Disk surface of a platter is divided into imaginary tracks and sectors. Tracks are concentric circles where the data is stored, and are numbered from the outermost to the innermost ring, starting with zero. There are about 50,000 to 100,000 tracks per platter and a disk generally has 1 to 5 platters. Tracks are further subdivided into sectors (or track sector). A sector is just like an arc that forms an angle at the center. It is the smallest unit of information that can be transferred to/from the disk. There are about hundreds of sectors per track and the sector size is typically 512 bytes. The inner tracks are of smaller length than the outer tracks, thus, there are about 500 sectors per track in the inner tracks, and about 1000 sectors per track towards the boundary. In general, the disk containing large number of tracks on each surface of platter and more sectors per track has higher storage capacity. A disk contains one read-write head for each surface of a platter, which is used to store and retrieve data from the surface of platter. Information is magnetically stored on a sector by the read-write head. The head moves across the surface of platter to access different tracks. All the heads are attached to a single assembly called a disk arm. Thus, all the heads of different platters move together. The disk platters mounted on a spindle together with the heads mounted on a disk arm is known as head-disk assemblies. All the read-write heads are on the equal diameter track on different platters at one time. The tracks of equal diameter on different platters form a cylinder. Accessing data of one cylinder is much faster than accessing data that is distributed among different cylinders. A close look at the internal structure of magnetic disk is shown in Figure 7.2. Learn More Some disks have one read-write head for each track of platter. These disks are termed as fixed-head disks, since one head is fixed on each track and is not moveable. On the other hand, disks in which the head moves along the platter surface are termed as moveable-head disks. Fig. 7.2 Moving head disk mechanism Accessing Data from Magnetic Disk Data in a magnetic disk is recorded on the surface of the circular tracks with the help of read/write head, which is mounted on the arm assembly. These heads can be multiple in numbers to access the adjacent tracks simultaneously and thus access to a disk is faster. The transfer of data between memory and disk drive is handled by a disk controller, which interfaces the disk drive to the computer system. Some common interfaces used for disk drives on personal computers and workstations are small-computer-system interconnect (SCSI) (pronounced "scuzzy"), AT attachment (ATA), and serial ATA (SATA). In the latest technology, disk 174

controller is implemented within the disk drive. The controller accepts high level I/O commands (to read or write sector) and start positioning the disk arm over the right track in order to read or write the data. Disk controller computes the checksums for the data to be written on the sector and attach it with the sector. When the sector is to be read, the controller again computes the checksum from the sector data and compares it with stored checksum. If there is any difference between them, the controller will retry reading data several times. However, if the difference between computed and stored checksum continues to occur, the controller signals a read failure. Remapping of bad sectors is another major task performed by the disk controllers. During the initial formatting of disk, if the controller detects a bad (or damaged) sector, it is logically mapped to another physical location by the disk controller. Disk is notified for the remapping and any further operation is carried out on the new location. The process of accessing data comprises three steps: 1. Seek: As soon as the disk unit receives the read/write command, the read/write heads are positioned on specific track on the disk platter. The time taken in doing so is known as seek time. It is average time required to move the heads from one track to some other desired track on the disk. Seek times of modern disk may range between 6–15 milliseconds. 2. Rotate: Once the heads are positioned on the desired track, the head of the specific platter is activated. Since the disk is rotated constantly, the head has to wait for the required sector or cluster (desired data) to come under it. This delay is known as rotational delay time or latency of the disk. The average rotational latencies range from 4.2 to 6.7 ms. 3. Data transfer: After waiting for the desired data location, the read/write head transfers the data to or from the disk to primary memory. The rate at which the data is read from or written to the disk is known as data transfer rate. It is measured in kilobits per second (kbps). Some of the latest hard disks have a data transfer rate of 66 MB/ second. The data transfer rate depends upon the rotational speed of the disk. If the disk has a rotational speed of 6000 rpm (rotations per minute), having 125 sectors and 512 bytes/sector, the data transfer rate per revolution will be 125 × 512 = 64000 bytes. Hence, the total transfer rate per second will be 64000 × 6000/60 = 6,400,000 bytes/ second or 6.4 MB/second. The combined time (seek time, latency time, and data transfer time) is known as the access time. Specifically, it can be described as the period of time that elapses between a request for information from disk or memory, and the information arriving at the requesting device. Memory access time refers to the time it takes to transfer a character from memory to or from the processor, while disk access time refers to the time it takes to place the read/write heads over the given sector and transfer the data to or from the requested device. RAM may have an access time of 80 nanoseconds or less, while disk access time could be 12–19 milliseconds. The reliability of the disk is measured in terms of mean time to failure (MTTF). It is the amount of time for which the system can run continuously without any failure. Manufacturers claim that the mean time to failure of disks ranges between 1,000,000 hours (about 116 years) to 1,500,000 hours (about 174 years). Although, various research studies conclude that failure rates are, in some cases, 13 times greater than what manufacturer claims. Generally, expected life span of most disks is about 4 to 5 years. However, disks have a high rate of failure when they become a few years old. 7.2 REDUNDANT ARRAYS OF INDEPENDENT DISKS The technology of semiconductor memory has been advancing at a much higher rate than the technology of secondary storage. The performance and capacity of semiconductor memory is much superior to secondary storage. To match this growth in semiconductor memory, a significant development is required in the technology of secondary storage. A major advancement in secondary storage technology is represented by the development of Redundant Arrays of Independent Disks (RAID). The basic idea behind RAID is to have a large array of small independent disks. Presence of multiple disks in the system improves the overall transfer rates, if the disks are operated in parallel. Parallelizing the operation of multiple disks allow multiple I/O to be serviced in parallel. This setup also offers opportunities for improving the reliability of data storage, because data can be stored redundantly on multiple disks. Thus, failure of one disk does not lead to loss of data. In other words, this large array of independent disks acts as a single logical disk with improved performance and reliability. Improving Performance and Reliability with RAID 175

In order to improve the performance of disk, a concept called data striping is used which utilizes parallelism. Data striping distributes the data transparently among N disks, which make them appear as a single large, fast disk. Striping of data across multiple disks improves the transfer rate as well, since operations are carried out in parallel. Data striping also balances the load among multiple disks. Learn More Originally RAID stands for Redundant Array of Inexpensive Disk, since array of cheap smaller capacity disk was used as an alternative to large expensive disk. Those days the cost per bit of data of smaller disk was less than that of larger disk. In the simplest form, data striping splits each byte of data into bits and stores them across different disks. This splitting of each byte into bits is known as bit-level data striping. Having 8-bits per byte, an array of eight disks (or either a factor or multiple of eight) is treated as one large logical disk. In general, bit i of each byte is written i th disk. However, if an array of only two disks is used, all odd numbered bits go to first disk and even numbered bits to second disk. Since each I/O request is accomplished with the use of all disks in the array, transfer rate of I/O requests goes to N times, where N represents the number of disks in the array. Alternatively, blocks of a file can be striped across multiple disks. This is known as block-level striping. Logical blocks of a file are assigned to multiple disks. Large requests for accessing multiple blocks can be carried out in parallel, thus improving data transfer rate. However, transfer rate for the request of a single block is same as it is in case of one disk. Note that the disks that are not participating in the request are free to carry out other operations. Having an array of N disks in a system improves the system performance; however, lowers the overall storage system reliability. The chance of failure of at least one disk out of total N disks is much higher than that of a specific single disk. Assume that the mean time to failure (MTTF) of a disk is about 150,000 hours (slightly over 16 years). Then, for an array of 100 disks, the MTTF of some disk is only 150,000/100 = 1500 hours (about 62 days). With such short MTTF of a disk, maintaining one copy of data in an array of N disks might result in loss of significant information. Thus, some solution must be employed to increase the reliability of such storage system. The most acceptable solution is to have redundant information. Normally, the redundant information is not needed; however, in case of disk failure it can be used to restore the lost information of failed disk. One simple technique to keep redundant information is mirroring (also termed as shadowing). In this technique, the data is redundantly stored on two physical disks. In this way every disk is duplicated, and all the data has two copies. Thus, every write operation is carried on both the disks. During read operation, the data can be retrieved from any disk. In case of failure of one disk, second disk can be used until the first disk gets repaired. If the second disk also fails before the repairing of first disk is completed, the data is lost. However, occurrence of such event is very rare. The mean time to failure of mirrored disk depends on two factors. 1. Mean time to failure of independent disks and 2. Mean time to repair a disk. It is the time taken, on an average, to restore the failed disk or to replace it, if required. Suppose failure of two disks is independent of each other. Further, assume that the mean time to repair of a disk is 15 hours and mean time to failure of single disk is 150,000 hours, then mean time to data loss in mirrored disk system is (150,000) 2 /(2 *15) = 7.5 * 10 8 hours or about 85616 years. An alternative solution to increase the reliability is storing error-correcting codes such as parity bits and hamming codes (discussed in next section). Such additional information is needed only in case of recovering the data of failed disk. Error- correcting codes are maintained in a separate disk called check disk. The parity bits corresponding to each bit of N disks are stored in check disk. 7.2.1 RAID Levels Data striping and mirroring techniques improve the performance and reliability of a disk. However, mirroring is expensive and striping does not improve reliability. Thus, several RAID organizations referred to as RAID levels have been proposed which aim at providing redundancy at lower cost by using the combination of these two techniques. These levels have 176

different cost-performance trade-offs. The RAID levels are classified into seven levels (from level 0 to level 6), as shown in Figure 7.3, and are discussed here. To understand all the RAID levels consider a disk array consisting of 4 disks. ? RAID level 0: RAID level 0 uses block-level data striping but does not maintain any redundant information. Thus, the write operation has the best performance with level 0, as only one copy of data is maintained and no redundant information needs to be updated. However, RAID level 0 does not have the best read performance among all the RAID levels, since systems with redundant information can schedule disk access based on shortest expected seek time and rotational delay. In addition, RAID level 0 is not fault-tolerant because failure of just one drive will result in loss of data. However, absence of redundant information ensures 100 per cent space utilization for RAID level 0 systems. ? RAID level 1: RAID level 1 is the most expensive system as this level maintains duplicate or redundant copy of data using mirroring. Thus, two identical copies of the data are maintained on two different disks. Every write operation needs to update both the disks, thus, the performance of RAID level 1 system degrades while writing. However, performance while read operation is improved by scheduling request to the disk with the shortest expected access time and rotational delay. With two identical copies of data, RAID level 1 system ensures only 50 per cent space utilization. ? RAID level 2: RAID level 2 is known as error-correcting code (ECC) organization. Two most popular error detecting and correcting codes are parity bits and hamming codes. In memory system, each byte is associated with a parity bit. The parity bit is set to 0 if the number of bits in the byte that are set to 1 is even; otherwise the parity bit is set to 1. If any one bit in the byte gets changed, then parity of that byte will not match with the stored parity bit. In this way, use of parity bit detects all 1-bit errors in the memory system. Hamming code has the ability to detect the damaged bit. It stores two or more extra bits to find the damaged bit and by complementing the value of damaged bit, the original data can be recovered. RAID level 2 requires three redundant disks to store error detecting and correcting information for four original disks. Thus, effective space utilization is about 57 per cent in this case. However, space utilization increases with the number of data disks because check disks grow logarithmically with the number of data disks. ? RAID level 3: As discussed, RAID level 2 uses check disks to hold information to detect the failed disk. However, disk controllers can easily detect the failed disk and hence, check disks need not to contain information to detect the failed disk. RAID level 3 maintains only single check disk with parity bit information for error correction as well as for detection. This level is also named as bit-interleaved parity organization. Performance of RAID level 2 and RAID level 3 are very similar but RAID level 3 has lowest possible overheads for reliability. So, in practice, level 2 is not used. In addition, level 3 has two benefits over level 1. Level 1 maintains one mirror disk for every disk, whereas level 3 requires only one parity disk for multiple disks, thus, increasing effective space utilization. In addition, level 3 distributes the data over multiple disks, with N-way striping of data, which makes the transfer rate for reading or writing a single block by N times faster than level 1. Since every disk has to participate in every I/O operation, RAID level 3 supports lower number of I/O operations per second than RAID level 1. ? RAID level 4: Like RAID level 0, RAID level 4 uses block-level striping. It maintains parity block on a separate disk for each corresponding block from N other disks. This level is also named as block-interleaved parity organization. To restore the block of the failed disk, blocks from other disks and corresponding parity block is used. Requests to retrieve data from one block are processed with only one disk, leaving remaining disks free to handle other requests. Writing a single block involves one data disk and check disk. The parity block is required to be updated with each write operation, thus, only one write operation can be processed at a particular point of time. With four data disks, RAID level 4 requires just one check disk. Effective space utilization for our example of four data disk is 80 per cent. However, as always one check disk is required to hold parity information, effective space utilization increases with the number of data disks. ? RAID level 5: Instead of placing data across N disks and parity information in one separate disk, this level distributes the block-interleaved parity and data among all the N+1 disks. Such distribution has advantage in processing read/write requests. All disks can participate in processing read request, unlike RAID level 4, where 177
dedicated check disks never participates in read request. So level 5 can satisfy more number of read requests in given amount of time. Since bottleneck of single check disk has been eliminated, several write request could also be processed in parallel. RAID level 5 has the best performance among all the RAID levels with redundancy. In our example of 4 actual disks, RAID level 5 has five disks overall, thus, effective space utilization for level 5 is same as in level 3 and level 4. ? RAID level 6: RAID level 6 is an extension of RAID level 5 and applies P + Q redundancy scheme using Reed- Solomon codes. Reed–Solomon codes enable RAID level 6 to recover from up to two simultaneous disk failures. RAID level 6 requires two check disks; however, like RAID level 5, redundant information is distributed across all disks using block-level striping. Fig. 7.3 Representing RAID levels 7.3 NEW STORAGE SYSTEMS Demand for storage of data is increasing across the organizations with the advent of Internet-driven applications such as e- commerce. Enterprise Resource Planning (ERP) systems and data warehouses need enough space to keep bulk data or information across the organization. In addition, the organizations have different branches across the world and providing data to different users across different branches in 24×7 environment is a challenging task. As a result, managing all the data becomes costly with an increase in simultaneous requests. In some cases, cost of managing server-attached storage exceeds the cost of the server itself. Therefore, various organizations choose the concept called storage area network (SAN) to manage their storage. 178

In SAN architecture, many server computers are connected with a large number of disks on a high-speed network. Storage disks are placed at a central server room, and are monitored and maintained by system administrators. The storage subsystem and the computer communicate with each other by using SCSI or fiber channel interfaces (FCI). Several SAN providers came up with their own topologies, thus, providing different performance and connectivity options allowing storage subsystem to be placed far apart from the server computers. Storage subsystem is shared among multiple computers that could process different parts of an application in parallel. Servers are connected to multiple RAID systems, tape libraries, and other storage systems in different configurations by using fiber-channel switches and fiber-channel hubs. Thus, the main advantage is many-to-many connectivity among server computers and storage system. Furthermore, isolation capability of SAN allows easy addition of new peripherals and servers. An alternative to SAN is network-attached storage (NAS). A NAS device is a server that allows file sharing. NAS does not provide any of the services that a typical server provides. NAS devices are being used for high performance storage solutions at low cost. NAS devices allow enormous amount of hard disk storage space to be made available to multiple servers without shutting them down for maintenance and upgrades. The NAS devices do not need to be located within the server, but can be placed anywhere in a local area network (LAN) and may be combined in different configurations. A single hardware device (called the NAS box or NAS head) acts as the interface between the NAS system and network clients. NAS units usually have a web interface and do not require a monitor, keyboard, or mouse. In addition, NAS system usually contains one or more hard disks or tape drives, often arranged into logical, redundant storage containers or RAID, as traditional file servers do. Further, NAS removes the file serving responsibility from other servers on the network. NAS provides both storage and file system. It is often contrasted with SAN, which provides only block-based storage and leaves file system concern on the client side. The file based protocols used by NAS are NFS or SMB, whereas SAN protocols are SCSI or fibre channel. NAS increases the performance as the file serving is done by the NAS and not by the server, which is responsible for doing other processing. The performance of NAS devices depends heavily on the speed of and traffic on the network, and also on the amount of cache memory on the NAS devices. 7.4 ACCESSING DATA FROM DISK The database is mapped into a number of different files, which are physically stored on disk for persistency and the underlying operating system is responsible for maintaining these files. Each file is decomposed into equal size pages, which is the unit of exchange between the disk and the main memory. The size of the page chosen is equal to the size of disk block and a page can be stored in any disk block. Hence, reading or writing a page can be done in one disk I/O. Generally, main memory is too small to hold all the pages of a file at one time. Thus, the pages of a file need to be transferred into main memory as and when requested by the CPU. If there is no free space in main memory, some existing page must be replaced to make space for the new page. The policy that is used to choose a page to be replaced with the new page is called the replacement policy. To improve the performance of a database, it is required to keep as many pages in the main memory as possible. Since it is not possible to keep all the pages in main memory at a time, the available space of main memory should be managed efficiently. The goal is to minimize the number of disk accesses by transferring the page in the main memory before a request for that page occurs in the system. The main memory space available for storing the data is called buffer pool and the subsystem that manages the allocation of buffer space is called buffer manager. The available main memory is partitioned into page size frames. Each frame can hold a page of file at any point of time. 7.4.1 Buffer Manager When a request for a page is generated, the buffer manager handles it by passing the memory address of the frame that contains the requested page to its requester, if the page is already in the buffer pool. However, if the page is not available in buffer pool, buffer manager allocates space for that page. If no free frame is available in buffer pool, allocation of space requires de-allocation of some other pages that are no longer needed by the system. After allocating the space in buffer pool, buffer manager transfers the requested page from disk to main memory. While de-allocating a page, the buffer manager writes back the updated information of that page on the disk, if the page has been modified since its last access from the disk (see Figure 7.4). 179

Fig. 7.4 Buffer management In addition to the buffer pool, the buffer manager maintains two variables for each frame in the buffer pool. ? pin_count: This variable keeps track of the number of times the page currently in a given frame has been requested but not released. In this way, it records the number of current users of the page. ? dirty: It is a Boolean variable that keeps track whether the current page in the frame has been modified or not, since it was brought into the frame from disk. Initially, the value of pin_count is set to 0 and dirty bit is turned off for every frame. On receiving a request for a page, the buffer manager first searches the buffer pool for that page. If the requested page is available in any frame, its pin_count is incremented by one. If no frame in the buffer pool is holding the requested page, buffer manager chooses a frame for replacement using replacement policy. If the dirty bit is on for the replacement frame, the page in that frame is written back to the disk and then, the requested page is transferred from disk to that frame in the buffer pool. After having the requested page in buffer pool, buffer manager passes the memory address of frame that contains the requested page to its requestor. The process of incrementing the pin_count is generally called pinning the page. When the requested page is subsequently released by the process, pin_count of that frame is decremented. It is called unpinning the page. At the time of unpinning the page, the dirty bit needs to be set by the buffer manager, if the page has been modified by its requestor. Note that the buffer manager will not replace the page in the frame until it is unpinned by all its requestor, that is, until its pin_count becomes 0. For each new page from disk, buffer manager always choose a frame with pin_count 0 to be replaced, if no free frame is available. In case two or more such frames are found, then buffer manager chooses a frame according to its replacement policy. 7.4.2 Page-Replacement Policies Buffer manager uses replacement policy to choose a page for replacement from the list of unpinned pages. The main goal of replacement policies is to minimize the number of disk accesses for the requested page. For general-purpose programs, it is not possible to anticipate accurately which page will be requested in near future and hence, operating system uses the pattern of past references to predict the future references. Commonly used replacement policies include least recently used (LRU), most recently used (MRU), and clock replacement. LRU replacement policy assumes that the pages that have been referenced recently have greater chance to be referenced again in the near future. Thus, the least recently referenced page should be chosen for replacement. However, the pattern of future references is more accurately predicted by the database system than an operating system. A user-request to the database system involves several steps and by carefully looking at these steps, database system can apply this policy more efficiently than operating system. In some cases, the optimal policy to choose a page for replacement is MRU policy. As the name indicates, MRU chooses the most recently accessed page for replacement, as and when required. Note that the page currently being used by the 180

system must be pinned and hence, they are not eligible for replacement. After completing the processing with the page, it must be unpinned by the system so that it becomes available for replacement. Another replacement policy is clock replacement policy, which is a variant of LRU. In this replacement policy, an additional reference bit is associated with each frame, which is set on as soon as pin_count of that frame becomes 0. All the frames are considered as arranged in a circular manner. A variable current moves around the logical circle of frames, starting with the value 1 through N (total number of frames in the buffer pool). The clock replacement policy considers the current frame for replacement. If the considered frame is not chosen for replacement, value of current is incremented by one and the next frame is considered. The policy continues to consider the frame until some frame is chosen for replacement. If the pin_count of current frame is greater than 0, then the current frame is not a candidate for replacement. If the current frame has pin_count 0 and reference bit on, the algorithm turns it off—this helps to ignore the recently referenced pages to be chosen for replacement. Only the frame having pin_count 0 and reference bit off can be chosen for replacement. If in some sweep of clock, all the frames have pin_count greater than 0, it means no frame in the buffer pool is a candidate for replacement. Learn More Buffer manager should not attempt to choose the pages of indexes (discussed in Section 7.7) for replacement, until or unless necessary, because pages of indexes are generally accessed much frequently than the pages of the file itself. Although, no single policy is suitable for all types of applications, a large number of database systems use LRU. The policy to be used by buffer manager is influenced by various factors other than time at which the page will be referenced again. If the database system is handling requests from several users concurrently, the system may need to delay certain requests by using some concurrency-control mechanism to maintain database consistency. Buffer manager needs to alter its replacement policy using information from concurrency control subsystem, indicating which requests are being delayed. 7.5 PLACING FILE RECORDS ON DISK BLOCKS A database consists of a number of relations where each relation is typically stored as a file of records. Each record of a file represents an entity instance and its attributes. For example, each record of PUBLISHER relation of Online Book database consists of P_ID, Pname, Address, State, Phone, and Email_id fields. These file records are mapped onto disk blocks. The size of blocks are fixed and is determined by the physical properties of disk and by the operating system; however, the size of records may vary. Generally in the relational database, different relations have tuple of different sizes. Thus, before discussing how file records are mapped onto disk blocks, we need to discuss how database is represented in terms of files. There are two approaches to organize the database in the form of files. In the first approach, all the records of a file are of fixed-length. However, in the second approach, the records of a file vary in size. A file of fixed-length records is simple to implement as all the records are of fixed-length. However, deletion of record results in fragmented memory. On the other hand, in case of files of variable-length records, memory space is efficiently utilized; however, locating the start and end of record is not simple. 7.5.1 Fixed-Length Records All the records in a file of fixed-length record are of same length. Consider a relation PUBLISHER whose record is defined here. create PUBLISHER as (P_ID char(5) PRIMARY KEY NOT NULL, Pname char(20) NOT NULL, Address char(50), State char(20), 181
Phone numeric(10), Email_id char(50)) It is clear from the given definition that each record consists of P_ID, Pname, Address, State, Phone, and Email_id fields, resulting in records of fixed-length, but of varying length fields. Assume that each character requires 1 byte and numeric(10) requires 5 bytes, then a PUBLISHER record is 150 bytes long. Figure 7.5 represents a record of PUBLISHER relation with starting of each field within the record. Fig. 7.5 Fixed-length record In a file of fixed-length records, every record consists of same number of fields and size of each field is fixed for every record. It ensures easy location of field values as their positions are pre-determined. Since each record occupies equal memory, as shown in Figure 7.6, identifying start and end of record is relatively simple. Fig. 7.6 Location of fixed-length records A major drawback of fixed-length records is that a lot of memory space is wasted. Since a record may contain some optional fields and space is reserved for optional fields as well—it stores null value if no value is supplied by the user for that field. Thus, if certain records do not have values for all the fields, memory space is wasted. In addition, it is difficult to delete a record as deletion of a record leaves blank space in between the two records. To fill up that blank space, all the records following the deleted record need to be shifted. It is undesirable to shift a large number of records to fill up the space made available by a deleted record, since it requires additional disk access. Alternatively, the space can be reused by placing a new record at the time of insertion of new records, since insertions tend to be more frequent. However, there must be some way to mark the deleted records so that they can be ignored during the file scan. In addition to simple marker on deleted record, some additional structure is needed to keep track of free space created by deleted or marked records. Thus, certain number of bytes is reserved in the beginning of the file for a file header. The file header stores the address of first marked record, which further points to second marked record and so on. As a result, a linked list of marked slot is formed, which is commonly termed as free list. Figure 7.7 shows the record of a file with file header pointing to first marked record and so on. New record is placed at the address pointed by file header and header pointer is changed to point next available marked record. In case, no marked record is available, new record is appended at the end of the file. 182

Fig. 7.7 Fixed-length records with free list of marked records 7.5.2 Variable-Length Records Variable-length records may be used to utilize memory more efficiently. In this approach, the exact length of field is not fixed in advance. Thus, to determine the start and end of each field within the record, special separator characters, which do not appear anywhere within the field value, are required (see Figure 7.8). Locating any field within the record requires scan of record until the field is found. Fig. 7.8 Organization of variable-length records with '%' delimiter Alternatively, an array of integer offset could be used to indicate the starting address of fields within a record. The i th element of this array is the starting address of the i th field value relative to the start of the record. An offset to the end of record is also stored in this array, which is used to recognize the end of last field. The organization is shown in Figure 7.9. For null value, the pointer to starting and end of field is set same. That is, no space is used to represent a null value. This technique is more efficient way to organize the variable-length records. Handling such an offset array is an extra overhead; however, it facilitates direct access to any field of the record. Fig. 7.9 Variable-length record organization using an array of field offsets Sometimes there may be possibility that the values for a large number of fields are not available or are null. In that case, we can store the sequence of pair &gt;field name, field value&lt; instead of just field values in each record (see Figure 7.10). In this figure, three separator characters are used—one for separating two fields, second one for separating field value from field name, and third one for separating two records. Fig. 7.10 Organization for variable-length record 183
It is clear from the discussion that the memory is utilized efficiently but processing the variable-length records require complicated program. Moreover, modifying the value of any field might require shifting of all the following fields, since new value may occupy more or less space than the space occupied by the existing value. 7.5.3 Mapping Records After discussing how the database is mapped to files, we will discuss how these file records can be mapped on to disk blocks. There are two ways to organize or place them on to disk blocks. Generally, multiple records are stored in one disk block; however, some files may have large records that cannot fit in one block. Block size is not always multiple of record size, therefore, each disk block might have some unused space. This unused space can be utilized by storing part of a record on one block and the rest on another. In case, consecutive blocks are not allocated to the file, pointer at the end of first block points to the block that contains the remaining part of its last record [see Figure 7.11(a)]. Such type of organization in which records can span more than one disk block is called spanned organization. Spanned organization can be used with variable-length records as well as with fixed-length records. On the other hand, in unspanned organization the records are not allowed to cross block boundaries, thus, part of each block is wasted [see Figure 7.11(b)]. Unspanned organization is generally used with fixed-length records. This organization ensures starting of records at known position within the block, which makes processing of records simple. It is advantageous to use spanned organization to reduce the lost space in each block if average record is larger than a block. Learn More In order to simplify the management of free space, most RDBMS impose an upper limit on the record size. However, now- a-days, database often needs to store image, audio, or video objects which are much larger than the size of disk block. In these situations, large objects are stored in separate file(s) and a pointer to them is stored in the record. NOTE Generally records are stored contiguously in the disk block that means next record starts where the previous record ends. Fig. 7.11 Types of record organization In case of fixed length record, equal number of records is stored in each block, thus, determining the start and end of record is relatively simple. However, in case of variable-length record, different number of records is stored in each block. Thus, some technique is needed to indicate the start and end of the record within each block. One common technique to implement variable-length records is slotted page structure, which is shown in Figure 7.12. Under this technique, some space is reserved for header in the beginning of each block that contains the following information. 184
? total number of records in the block ? end of free space in the block ? location and size of each record within the block Fig. 7.12 Slotted page structure In a slotted page structure starting from the end of the block, the actual records are stored contiuously and the free space is contiguous between the final entry in the header array and the first record. The space for new record is allocated at the end of free space and corresponding entry of record location and record size is included in the header. Record is deleted by removing its entry from the header and freeing the space occupied by the record. To make the free space contiguous within the block again, all the records before deleted records are shifted to their right. In addition, header entry pointing to the end of free space is updated as well. Here, no outside pointers point directly to actual records, rather they must point to the entry in the header, which contains the location and size of each record. This support of indirect pointers to records allows records to be moved within the block to prevent fragmented free space inside a block. 7.6 ORGANIZATION OF RECORDS IN FILES Arrangement of the records in a file plays a significant role in accessing them. Moreover, proper organization of files on disk helps in accessing the file records efficiently. There are various methods (known as file organization) of organizing the records in a file while storing a file on disk. Some popular methods are heap file organization, sequential file organization, and hash file organization. 7.6.1 Heap File Organization It is the simplest file organization technique in which no particular order of record is maintained in the file. Instead, records

100%	MATCHING BLOCK 159/319	SA	U234.pdf (D109498494)
can be placed anywhere in the file, where there is			

enough space for that record. Such an organization is generally termed as heap file or pile file. Heap file is divided into pages of equal size and every record is identified by its unique id or record id. Operations that can be carried out on a heap file include creation and deletion of files, insertion and deletion of record, searching a particular record, and scanning of all records in the file. Heap file supports efficient insertion of new record; however, scanning, searching, and deletion of records is an expensive process. Generally, new record is appended at the end of file. The last page of file is copied to the buffer, new record is added and then the page is written back to the disk. The address of last page of file is kept in the file header. Scanning requires retrieving all the pages of heap file and processing each record on the page one after the other. Let a file has R records on each page and every record takes S time to process. Further, if the file contains P pages and retrieving one page requires T time, then total cost to scan the complete file is P(T+RS). Assuming that exactly one record satisfies the equality selection, it means, the selection is specified on a candidate key whose values are uniformly distributed. On an average, half the file needs to be scanned using linear search technique. If no record satisfies the given condition then, all the records of the file need to be scanned to verify that the searched record is not found. 185
For deleting a particular record, it needs to be located in the file using its record-id. Record-id helps in identifying the page that contains the record. The page is then transferred in the buffer and after having located the record it is removed from the page and the updated page is written back. Implementing Heap Files As stated earlier, heap files support various operations. To implement deletion, searching, and scanning operation, there is a need to keep track of all the pages allocated to that file. However, for insertion operation, identification of the pages that contain free space is also required. A simple way to keep this information is to maintain two linked lists of pages—one for those having no free space and another for those having some free space. The system only needs to keep track of first page of the file called header page. One such representation of heap file is shown in Figure 7.13. Fig. 7.13 Linked list of heap file organization When a new record is to be inserted in the file, the page with enough free space is located and the record is inserted in that page. If there is no page with enough space to accommodate that new record, a new page is allocated to the file and record is inserted in the new page. The main disadvantage of this technique is that we sometimes may require examining several pages for inserting a new record. Moreover, in case of variable length records, each page might always have some free space in it. Thus, all the pages allocated to the file will be on the list of free pages. This disadvantage of implementing heap file can be addressed by another technique in which a directory of pages is maintained. In this directory, each entry has a pointer to a page in the file and the amount of free space in that page. When directory itself contains several pages, they are linked together to form a linked list. Organization of heap file using directory is shown in Figure 7.14. 186
Fig. 7.14 Directory-based heap file organization 7.6.2 Sequential File Organization Often, it is required to process the records of a file in the sorted order based on the value of one of its field. If the records of the file are not physically placed in the required order, it consumes time to fulfill this request. However, if the records of that file are placed in the sorted order based on that field, we would be able to efficiently fulfill this request. A file organization in which records are sorted based on the value of one of its field is called sequential file organization, and such a file is called sequential file. In a sequential file, the field on which the records are sorted is called ordered field. This field may or may not be the key field. In case, the file is ordered on the basis of key, then the field is called the ordering key. Searching of records is more efficient in a sequential file if search condition is specified on the ordering field because then binary search is applicable instead of linear search. Moreover, retrieval of records with the range condition specified on the ordering field is also very efficient. In this operation, all the records starting with the first record satisfying the range selection condition till the first record that does not satisfy the condition are retrieved. NOTE Sequential file does not make any improvement in processing the records in random order. However, handling deletion and insertion operations are complicated. Deletion of a record leaves a blank space in between the two records. This situation can be handled by using deletion marker in the same way as already discussed in the Section 7.5.1. When a new record is to be inserted in a sequential file, there are two possibilities. First, we can insert the record at its actual position in the file. Obviously, this needs locating the first record that has to come after the new record and making space for the new record. Making space for a record may require shifting a large number of records and this is very costly in terms of disk access. Second, we can insert that record in an overflow area allocated to the file instead of its correct position in the original file. Note that the records in the overflow area are unordered. Periodically, the records in the overflow area are sorted and merged with the records in the original file. The second approach of insertion makes the insertion process efficient; however, it may affect the search operation. This is because the required record needs to be searched in the overflow area using linear search if it is not found in the original file using binary search. 187

7.6.3 Hash File Organization In this organization, records are organized using a technique called hashing, which enables very fast access to records on the basis of certain search conditions. This organization is called hash file or direct file. In hashing, a hash function h is applied to a single field, called hash field, to determine the page to which the record belongs. The page is then searched to locate the record. In this way, only one page is accessed to locate a record. Note that the search condition must be an equality condition on the hash field. The hash field is known as hash key in case the hash field is the key field of the file. Hashing is typically implemented as a hash table through the use of an array of records. Given the hash field value, the hash function tells us where to look in the array for that record. Suppose that there are N locations for records in the array, it means the index of array ranges from 0 to N–1. The hash function h converts the hash field value between 0 and N–1. Note that non-integer hash field value can be converted into integer before applying hash function. For example, the numeric (ASCII) code associated with characters can be used in converting character values into integers. Some common hash functions are discussed here. Things to Remember Locating a record for equality search conditions is very effective using hashing technique. However, with range conditions, it is almost as worse as scanning all the records of the file. ? Cut key hashing: This hash function 'cuts' some digits from the hash field value (or simply key) and uses it as hash address. For example, if there are 100 locations in the array, a 2-digit address, say 37, may be obtained from the key 1324 37 by picking the highlighted digits. It is not a good algorithm because most of the key is ignored and only a part of the key is used to compute the address. ? Folded key: This hash function involves applying some arithmetic functions, such as addition/ subtraction or some logical functions, such as and/or to the key value. The result obtained is used as the record address. For example, the key is broken down into a group of 2-digit numbers from the left most digits and they are added like 13 + 24 + 37 = 74. The sum is then used as record address. It is better than cut key hashing as the entire key is used but not good enough as records are not distributed evenly among pages. ? Division-remainder hashing: This hash function is commonly used and in this the value of hash field is divided by N and remainder is used as record address. Record address = (Key value) mod (N). For example, (132437) mod (101) = 26 This technique works very well provided that N is either a prime number or does not have a small divisor. The main problem associated with most hashing functions is that they do not yield distinct addresses for distinct hash field values, because

89%	MATCHING BLOCK 160/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

the number of possible values a hash field can take is much larger than the number of available addresses

to store records. Thus, sometimes a problem, called

71%	MATCHING BLOCK 161/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

collision, occurs when hash field value of a record to be inserted hashes to an address

which is already being occupied by another record. It should be resolved by finding some other location to place the new record. This process of finding another location is called collision resolution. Some popular methods for collision resolution are given here. ? Open addressing: In this method, the program keeps checking the subsequent locations starting from the occupied location specified by hash function until an unused or empty location is found. ? Multiple hashing: In this method, the program can use another hash function if the first hash function results in a collision. If another collision occurs, the program may apply another hash function and so on, or it can use open addressing if necessary. This technique of applying multiple hash function is called double hashing or multiple hashing. ? Chained overflow: In this method, all the records whose addresses are already occupied are stored in an

71%	MATCHING BLOCK 163/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

overflow space. In addition, a pointer field is associated with each record location. A collision is resolved by placing the new record in overflow space and the pointer field of occupied hash address

location is set to that location in overflow space. In this way, a linked list of records, which hash to same address, is maintained, as shown in Figure 7.15. 188
Fig. 7.15 Collision resolution using chained overflow All these methods work well; however, each of them

100%	MATCHING BLOCK 164/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

requires its own algorithms for insertion, retrieval, and deletion of records.

All the algorithms for chained overflow are simple, whereas open addressing requires tricky algorithm for deletion of records. A good hash function must distribute records uniformly over the available addresses so as to minimize collisions. In addition, it should always return same value for same input. Generally, hashing works best when division remainder hash function is used in which N is chosen as a prime number, as it distributes the records better over the available addresses. Further, the collision problem is less severe by using the concept of buckets, which represents the storage unit that can store one or more records. There are two types of hashing schemes namely, static hashing and dynamic hashing, depending on the number of buckets allocated to a file. Static Hashing The hashing scheme in which a fixed number of buckets, say N, are allocated to a file to store records is called static hashing. The pages of a file can be viewed as

95%	MATCHING BLOCK 162/319	W

a collection of buckets, with one primary page and additional overflow pages.

The file consists of 0 to N–1 buckets, with one primary page per bucket initially. Further, let V denote the set of all values hash field can take then the hash function h maps V to N. To insert a record, hash function is applied on the hash field to identify the bucket to which the record belongs and then it is placed there. If the bucket is already full, a new overflow page is allocated and the record is placed in the new overflow page and then the page is added to the overflow chain of bucket. To search a record, the hash function transforms the hash field value to the bucket to which the required record belongs. Hash field value of all the records in the bucket is then verified to search the required record. Note that to speed up the searching process within the bucket, all the records are maintained in a sorted order by hash field value. 189
To delete a record, hash function is applied to identify the bucket to which it belongs. The bucket is then searched to locate the corresponding record and after locating the record, it is removed from the bucket. Note that the overflow page is removed from the overflow chain of the bucket if it is the last record in the overflow page. Figure 7.16 illustrates the static hashing which represents the records of BOOK relation with Price as hash field. In this example, the hash function h takes the remainder of Price after dividing it by three to determine the bucket for a record. Function h is defined as h(Price) = 0, record for bucket 1 h(Price) = 1, record for bucket 2 h(Price) = 2, record for bucket 3 Fig. 7.16 Static hashing with Price as hash field In static hashing, since the number of buckets allocated to a file are fixed when the file is created, the primary pages can be stored on consecutive disk pages. Hence, searching a record requires just one disk I/O, and other operations, like insertion and deletion, require two I/Os (read and write the page). However, as the file grows, long overflow chains are developed which degrade the performance of the file as searching a bucket requires searching all the pages in its overflow chain. It is undesirable to use static hashing with files that grow or shrink a lot dynamically, since number of buckets is fixed in advance. It is a major drawback for dynamic files. Suppose a file grows substantially more than the allocated space, a long overflow chain is developed which results in poor performance. Similarly, if a file shrinks greatly, a lot of space is left unused. In either case, the number of buckets allocated to the file needs to be changed dynamically and new hash function based on new value of N should be used for distribution of records. However, such reorganization consumes a lot of time for large files. Another alternative is to use dynamic hashing, which allows number of buckets to vary dynamically with only minor (internal) reorganization. Dynamic Hashing As discussed earlier, the number of available addresses for records is fixed in advance in static hashing, which makes it unsuitable for the files that grow or shrink dynamically. Thus, dynamic hashing technique is needed which allocates new buckets dynamically as needed. In addition, it allows

96%	MATCHING BLOCK 165/319	SA	248E1130_RDBMS.docx (D165247738)

the hash function to be modified dynamically to accommodate the growth or shrinkage of database

files. Dynamic hashing is of two forms, namely, extendible hashing and linear hashing. In our discussion of dynamic hashing, we consider the result of hash function as binary representation of hash field value. 190

Extendible Hashing Extendible hashing uses a directory of pointers to buckets. A directory is just an array of 2 d size, where d (called global depth) is the number of bits of hash value used to locate the directory element. Each element of the directory is a pointer to a bucket in which the corresponding records are stored. In this technique, the hash function is applied on the hash field value and the last d bits of hash value are used to locate the directory element. The pointer in this array position points to the bucket to which the corresponding record belongs. To understand this technique, consider a directory consisting of an array of 2 2 size (see Figure 7.17), which means an array of size 4. Here d is 2, thus, last 2 bits of hash value are used to locate a directory element. Further, assume that each bucket can hold three records. The search for a key, say 21, proceeds as follows. Last 2 bits of hash value 21 (binary 10101) hashes into the directory element 01, which points to bucket 2. Fig. 7.17 Extendible hashed file Consider the insertion of a new record with hash field value 9 (binary 1001). It hashes to the directory element 01, which further points to the bucket 2. Since bucket 2 has space for a new record, it is inserted there. Next, consider the insertion of a record with hash field value 24 (binary 11000). It hashes to the directory element 00 which points to bucket 1, which is already full. In this situation, extendible hashing splits this full bucket by allocating a new bucket and redistributes the records across the old bucket and its split image. To redistribute the records among these two buckets, the last three bits of hash value are considered. The last two bits (00 in this case) indicate the directory element and the third bit differentiate between these two buckets. Splitting of bucket with redistributing of records is shown in Figure 7.18. Now, the contents of bucket 1 and bucket 1a are identified by 3 bits of hash value, whereas the contents of all other buckets can be identified by 2 bits. To identify

100%	MATCHING BLOCK 166/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

the number of bits on which the bucket contents are based,

a local depth is maintained with each bucket, which specifies the number of bits required to identify its contents. 191 Fig. 7.18 Extendible hashing with splitting bucket Note that if the local depth of overflowed bucket is equal to the global depth there is a need to double the number of entries in the directory and global depth is incremented by one. However, if local depth of overflowed bucket is less than the global depth there is no need to double the directory entries. For example, consider the insertion of a record with hash value 5 (binary 101). It hashes to the directory element 01 which points to bucket 2, which is already full. This situation is handled by splitting the bucket 2 and using the directory element 001 and 101 to point to the bucket 2 and its split image, respectively, (see Figure 7.19). Fig. 7.19 After inserting record with hash value 5 To delete a record, first the record is located and then removed from the bucket. In case, the deletion leaves the bucket empty the bucket can be merged with its split bucket image. The local depth is decreased if the buckets are merged. After 192

merging, if the local depth of all the buckets becomes less than the global depth, the number of entries in the directory can be halved and global depth is be reduced by one. The main advantage of extendible hashing over static hashing is that the performance of the file does not degrade as the file grows dynamically. In addition, there is no space allocated in advance for future growth of the file. However, buckets can be allocated dynamically as needed. The size of directory is likely to be much smaller than the file itself, since each directory element is just a page-id. Thus, the space overhead for the directory table is negligible. The directory can be extended up to 2 n , where n is the number of bits in the hash value. Another advantage of extendible hashing is that bucket splitting requires minor reorganization in most cases. The reorganization is expensive when directory needs to be doubled or halved. One disadvantage of extendible hashing is that the directory must be accessed and searched before accessing the record in the bucket. Thus, most record retrievals require two block accesses, one for the directory and the other for the bucket. Linear Hashing Linear hashing also allows a hash file to grow or shrink dynamically by allocating new buckets. In addition, there is no need to maintain a directory of pointers to buckets. Like static hashing, collision can be handled by maintaining overflow chains of pages; however, linear hashing solves the problem of long overflow chains. In linear hashing, overflow of any bucket in the file leads to a bucket split. Note that the bucket to be split is not necessarily the same as the overflow bucket, instead buckets are chosen to be split in the linear order 0, 1, 2, ... To understand the linear hashing scheme, consider a file with T buckets, initially numbered 0 through T–1. Suppose that the file uses a mod hash function h i , denoted by h(V) = V mod T. Linear hashing uses a value j to determine the bucket to be split. Initially j is set to 0 and is incremented by 1 each time a split occurs. Thus, whenever an insert triggers an overflow record in any bucket, the j th bucket in the file is split into two buckets, namely, the bucket j and a new bucket T + j and j is incremented by 1. In addition, all the records of original bucket j are redistributed among the bucket j and the bucket T + j using a new hash function h i+1 , denoted by h i+1 (V) = V mod 2T. Thus, when all the original T buckets have been split, all buckets use the hash function h i+1 . A key property of hash function h i+1 is that any record that is hashed to bucket j by h i will hash either to bucket j or bucket T + j. The range of hash function h i+1 is twice as that of h i , that is, if h i hashes a record into one of T buckets, h i+1 hash that record in one of 2T buckets. In order to search a record with hash key value V, first the hash function h i is applied to V and if h i (V) &gt; j, it means that the hashed bucket is already split. Then, the hash function h i+1 is applied to V to determine which of the two split buckets contain the record. Since j is incremented with each split, so, when j = T, it indicates that all the original buckets have been split. At this point, j is reset to 0 and any overflow leads to the use of new hash function h i+2 , denoted by h i+2 (V) = V mod 4T. In general, linear hashing scheme uses a family of hash functions h i+k (V) = V mod (2 k T), where k = 0, 1, 2, ... Each time when all the original buckets 0 through (2 k T) - 1 have been split, k is incremented by 1 and new hashing function h i+k is needed. 7.7 INDEXING Once the records of a file are placed on the disk using some file organization method, the main issue is to provide quick response to different queries. For this, additional structure called index on the file can be created. Creating an index on a file does not affect the physical placement of the records but still provides efficient access to the file records. Index can be created on any field of the file, and that field is called indexing field or indexing attribute. Further, more than one index can be created on a file. Accessing records of a file using index requires accessing the index, which helps us to locate the record efficiently. This is because the values in the index are stored in the sorted order and the size of the index file is small as compared to the original file. Thus, applying binary search on index is efficient as compared to applying binary search on the original file. Different types of indexes are possible, which we discuss in this section. 7.7.1 Single-Level Indexes 193
Indexes can be created on the field based on which the file is ordered as well as on the field based on which the file is not ordered. Consider an index created on the ordered attribute ISBN of the BOOK relation (see Figure 7.20). This index is called primary index, since it is created on the primary key of the relation. Fig. 7.20 Index on the attribute ISBN of BOOK relation This index file contains two attributes—first attribute stores the value of the attribute on which the index is created and second attribute contains a pointer to the record in the original file. Note that it contains an entry for the first record in each disk block instead of every value of the indexing attribute. Such types of indexes which do not include an entry for each value of the indexing attribute are called sparse indexes (or non-dense indexes). On the other hand, the indexes which include an entry for each value of the indexing attribute are called dense indexes. Given the search-key value for accessing any record using primary index, we locate the largest value in the index file which

100%	MATCHING BLOCK 168/319	SA	248E1130_RDBMS.docx (D165247738)
is less than or equal to the search-key value.			

The pointer corresponding to this value directs us to the first record of the disk block in which the required record is placed (if available). If the first record is not the required record, we sequentially scan the entire disk block in order to find that record. Insertion and deletion operation on the ordered file is handled in the same way as discussed in the sequential file. Here; however, it may also require modifying the pointers in several entries of the index file. It is not always the case that index is created on the primary key attribute. If an index is created on the ordered non-key attribute, then it is called clustering index and that attribute is called clustering attribute. Figure 7.21 shows a clustering index created on the non-key attribute P_ID of the BOOK relation. Learn More Though dense index provides faster access to the file records than sparse index, having a sparse index with one entry per block is advantageous in terms of disk space and maintenance overhead while update operations. Moreover, once the disk block is identified and its records are brought into the memory, scanning them is almost negligible. 194
Fig. 7.21 Clustering index on P_ID field of BOOK relation This index contains an entry for each distinct value of the clustering attribute and the corresponding

pointer points to the first record with that clustering attribute value. Other records with the same clustering attribute value are stored sequentially after

that record, since the file is ordered on that attribute. Inserting a new record in the file is easy if space is available in the block in which it has to be placed. Otherwise, a new block is allocated to the file and the new record is placed in that block. In addition, the pointer of the block to which the record actually belongs is made to point to the new block. So far, we have discussed the case in which the index is created on the ordered attribute of the file. Indexes can also be created on the non-ordered attribute of the file. Such indexes are called secondary indexes. The indexing attribute may be a candidate key or a non-key attribute with duplicate values. In either case, secondary index must contain an entry for each value of the indexing attribute. It means the secondary index must be a dense index. This is because the file is not ordered on the indexing attribute and if some of the values are stored in the index, it is not possible to find a record whose entry does not exist in the index. Figure 7.22 shows a secondary index created on the non-ordered attribute Book_title of Book relation. 195
Fig. 7.22 Secondary index with record pointers on a non-ordering key field of BOOK relation This index contains an entry for each value of the indexing attribute and the corresponding pointer points to the record in the file. Note, that the index file itself is ordered on the indexing attribute. Inserting and deleting a record is straightforward with a little modification in the index file. If the indexing attribute of the secondary index is a non-key attribute with duplicate values, then we may use an extra level of indirection to handle multiple pointers. In this case, the pointer in the index points to a bucket of pointers instead of the actual records in the file. Each pointer in the bucket either points to the records in the file or to the disk block that contains the record. This structure of secondary index is illustrated in Figure 7.23. Fig. 7.23 Secondary index on a non-key field using an extra level of indirection 196
In general, secondary index occupies more space and consumes more time in searching, since it contains more entries than the primary index. However, having a secondary index is advantageous because in the absence of secondary index, accessing any random record requires a linear scan of file. On the other hand, if primary index does not exist, binary search is still applicable on the file, since records are ordered. 7.7.2 Multilevel Indexes It is common to have several thousands (or even lacs) of records in the database of medium or large-scale organizations. As the size of the file grows, the size of the index (even sparse index) grows as well. If the size of the index becomes too large to fit in the main memory at once, it becomes inefficient for processing, since it must be kept sequentially on disk just like an ordinary file. It implies that searching large indexes require several disk-block accesses. One solution to this problem is to view the index file, which we refer to as the first or base level of a multilevel

index, just as any other sequential file and construct a primary index on the index

file. This index to the first level is called the second level of the multilevel index. In order

to search a record, we first apply binary search on the second level index to find the largest value which is

48%	MATCHING BLOCK 171/319	SA	248E1130_RDBMS.docx (D165247738)

less than or equal to the search-key. The pointer corresponding to this value points to the block of the first level index that contains

an entry for the searched record. This block of first level index is searched to locate the largest value which is

48%	MATCHING BLOCK 172/319	SA	248E1130_RDBMS.docx (D165247738)

less than or equal to the search-key. The pointer corresponding to this value directs us to the block of file that contains the required record.

If second level index is too small to fit in main memory at once, fewer blocks of index file needs to be accessed from disk to locate a particular record. However, if the second level index file is too large to fit in main memory at once, another level of index can be created. In fact, this process of creating the index on index can be repeated until the size of index becomes small enough to fit in main memory. This type of index with multiple levels (two or more levels) of index is called multilevel index. Figure 7.24 illustrates multilevel indexing. Fig. 7.24 Multilevel indexing reduces the number of disk accesses while searching for a record; however, insertion and deletion are complicated because all the index files at different levels are ordered files. Any insertion or deletion may cause several index files at different levels to be modified. One solution to this problem is to keep some space reserved in each block for new entries. This is often implemented using B-trees and B + -trees. B-trees 197
It is a kind of tree satisfying some additional properties or constraints. Each node of a B-tree consists of n−1 values and n pointers. Pointers can be of two types, namely, tree pointers and data pointers. The tree pointer points to any other node of the tree and the data pointer points either to the block which contains the record with that index value or to the record itself. The order of information within each node is &gt;P 1 , &gt;V 1 , R 1 &lt;, P 2 , &gt;V 2 , R 2 &lt;, P 3 , &gt;V 3 , R 3 &lt;, ..., P n−1 , &gt;V n−1 , R n−1 &lt;, P n &lt; where P i is a tree pointer, V i is any value of indexing attribute and R i is a data pointer (see Figure 7.25). Fig. 7.25 A node of B-tree NOTE The structure of leaf node and internal node are same except the difference that all the tree pointers in a leaf node are null. All the values in a node should be in the sorted order, it means V 1 &gt; V 2 &gt; ··· &gt; V n−1 within each node of the tree. In addition, the node pointed to by any tree pointer P i should contain all the values larger than the value V i−1 and less than the value V i . Another constraint that the B-tree must satisfy is that it must be balanced; it means all the leaf nodes should be at the same level. B-tree also ensures that each node, except root and leaf nodes, should be at least half full, that means it should have at least n/2 tree pointers. The number of tree pointers in each node (called fan-out) identifies the order of B-tree. Maintaining all the constraints of a B-tree while insertion and deletion operation requires complex algorithms. First, consider the insertion of a value in a B-tree. Initially, the B-tree has only one node (root node) which is a leaf node at level 0. Suppose that the order of the B-tree is n. Thus, when n−1 values are inserted into the root node, it becomes full. Next insertion splits the root node into two nodes at level 1. The middle value, say m, is kept in the root node and the values from 1 to m−1 are shifted in the left child node and values from m+1 to n−1 are shifted in the right child node. However, when such situation occurs with a non-root node, it

63%	MATCHING BLOCK 173/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

is split into two nodes at the same level and middle value is transferred to its parent node.

Parent node then points to both the nodes as its left child and right child. If the parent node is also full, it is also split and splitting may propagate to other tree levels. If splitting gets propagated all the way to the root node and if root is also split, it increases the number of levels of the B-tree. Although, insertion in a node that is not full is quite simple and efficient. A typical B-tree of order n = 3 is shown in Figure 7.26. Fig. 7.26 A B-tree of order n=3 Now, consider the deletion of a value from a B-tree. If deleting a value from a node leaves the node more than half empty, its values are merged with its neighbouring nodes. This may also propagate all the way to the root of the B-tree. Thus, deletion of a value may result in a tree with lesser number of levels than the number of levels of the tree before deletion. However, deletion from a node that is full or nearly full is a simple process. B + -trees B + -tree, a variation of B-tree, is the most widely used structure for multilevel indexes. Unlike B-tree, B + -tree includes all the values of indexing attribute in the leaf nodes, and internal nodes contain values just for directing the search operation. All the pointers in the internal nodes are tree pointers and there is no data pointer in them. All the data pointers are included in the leaf nodes and there is one tree pointer in each leaf node that points to the next leaf node. Structure of leaf node and internal nodes of a B + -tree is shown in Figure 7.27. 198

Fig. 7.27 Nodes of a B + -tree Each leaf node has n−1 values along with a data pointer for each value. If the indexing attribute is the key attribute, the data pointer points either to the record in the file or to the disk block containing the record with that indexing attribute value. On the other hand, if the indexing attribute is not the key attribute, the data pointer points to the bucket of pointers, each of which points to the record in the file. The leaf nodes and internal nodes of a B + -tree corresponds to the first level and other levels of multilevel index, respectively. Since data pointers are not included in the internal nodes of a B + -tree,

76%	MATCHING BLOCK 174/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

more entries can be fit into an internal node of a B + - tree than corresponding B-tree.

As a result, we have larger order for B + -tree than B-tree for the same size of disk block. A typical B + -tree of order n=3 is shown in Figure 7.28. Fig. 7.28 A B + -tree of order n=3 In B-tree and B + -tree data structure, each node corresponds to a disk block. In addition, each node is kept between half-full or completely full. Indexed Sequential Access Method A variant of B + -tree is indexed sequential access method (ISAM). In ISAM, the leaf nodes and the overflow nodes (chained to some leaf nodes) contain all the values of indexing attribute and data pointers, and the internal nodes contain values and tree pointers just for directing the search operation. Unlike B + -tree, the ISAM index structure is static, it means the number of leaf nodes never change; if need arises overflow node is allocated and chained to the leaf node. The leaf nodes are assumed to be allocated sequentially, thus, no pointer is needed in the leaf node to point to the next leaf node. The algorithms for insertion, deletion, and search are simple. All searches start at root node and the value in the node helps in determining the next node to search. To insert a value, first the appropriate node is determined and if there is space in the node, the value is inserted there. Otherwise, an over-flow node is allocated and chained to the node, and then the value is inserted in the overflow node. To delete a value, the node containing the value is determined and the value is deleted from that node. In case, it is the only value in the overflow node, the overflow node is also removed. In case, it is 199

the only value in the leaf node, the node is left empty to handle future insertions (data from overflow nodes, if any, linked to that leaf node is not moved to the leaf node). The ISAM structure provides the benefits of sequentially allocated leaf nodes and fast searches as in the case of B + -trees. However, the structure is unsuitable for situations where file grows or shrinks much. This is because if too much insertions are made to the same leaf node, a long overflow chain can develop, which degrades the performance while searching as overflow nodes need to be searched if the search reaches to that leaf node. A sample ISAM tree is shown in Figure 7.29. Fig. 7.29 Sample ISAM tree 7.7.3 Indexes on Multiple Keys So far, we have discussed that the index is created on primary or secondary keys, which consists of single attributes. However, many query requests are based on multiple attributes. For such queries, it is advantageous to create an index on multiple attributes that are involved in most of the queries. To understand the concept, consider this query: List all the books from BOOK relation where Category = "Textbook" and P_ID = "P001". This query involves two non-key attributes, namely, Category and P_ID, thus, multiple records may satisfy either condition. There are three possible strategies to process this query, which are given here. ? Assume that we have an index on Category field. Using this index we can find all the books having Category = "Textbook". From these records, we can search all those books whose P_ID = "P001". ? Assume that we have an index on P_ID field. Using this index we can find all the records having P_ID = "P001". From these records, we can search all those books whose Category = "Textbook". ? Assume that we have an index on both the attributes, namely, Category and P_ID. Using index on Category field, we can find the set of all the books whose Category = "Textbook". Similarly, using index on P_ID field we can find all the books whose P_ID = "P001". By performing the intersection of both the set, we can find all the books satisfying both the given conditions. All of the strategies yield a correct set of records; however, if many records satisfy individual condition and only a few records satisfy both the conditions, then none of the above alternative is a good choice in terms of efficiency. Thus, some efficient technique is needed. There are a number of techniques that would treat the combination of Category and P_ID as a search-key. One solution to this problem is an ordered index on multiple attributes. The idea behind ordered index on multiple attribute is to create an index on combination of more than one attribute. It means search-key consists of multiple attributes. Such a search-key containing multiple attribute is termed as composite search-key. In this example, search-key consists of both the attributes, namely, Category and P_ID. In this type of index, search-key represents a tuple with values &gt;V 1 , V 2 , V 3 , ..., V n &lt; where index attributes are &gt;A 1 , A 2 , A 3 , ..., A n &lt;. The values of search-key are lexicographic ordered. Lexicographic ordering for above case states that all the entries of books published by P001 precedes the books published by P002 and so on. This type of index differs from the other index only in terms of search-key. However, basic structure is similar as that of any 200
other index having search-key on single attribute. Thus, working of index on composite search-key is similar as any other index discussed so far. Another solution to this problem is partitioned hashing, an extension of static hashing, which allows access on multiple keys. Partitioned hashing does not support range queries; however, it is suitable for equality comparisons. In partitioned hashing, for a composite search-key consisting of n attributes, the chosen hash function produces n separate hash addresses. These n addresses are concatenated to obtain the bucket address. All the buckets, which match the part of the address, are then searched for the combined search-key. The main advantage of partitioned hashing is that it can be easily extended to any number of attributes in the search-key. In addition, there is no need to maintain separate access structure for individual attributes. On the other hand, the main disadvantage of partitioned hashing is that it does not support range queries on any of the component attributes. Another alternative to such type of a problem is grid files. In the grid files, a grid array with one linear scale (or dimension) for each of the search attribute is created. Linear scales are made in order to provide uniform distribution of that attribute values in the grid array. Linear scale groups the values of one search-key attribute to each dimension. Figure 7.30 shows a grid array along with two linear scales, one is for attribute Category and the other is for attribute P_ID. Each grid cell points to some bucket address where the corresponding records are stored. The number of attributes in search-key determines the dimension of grid array. Thus, for search-key having n attributes, the grid array would have n dimensions. In our example, two attributes are taken as search-key, thus, the grid array is of two dimensions. Linear scale is looked up to map into the corresponding grid cell. Thus, the query for Category = "Textbook" and P_ID = "P001" maps into the cell (2, 0), which further points to the bucket address where the required records are stored. The main advantage of this method is that it can be applied for range queries. In addition, it is extended to any number of attributes in the search-key. Grid file method is good for multiple-key search. On the other hand, the main disadvantage is that it represents space and management overhead in terms of grid array because it requires frequent reorganization of grid files with dynamic files. Fig. 7.30 Grid array along with two linear scales SUMMARY 1. The database is stored on storage medium in the form of files, which is a collection of records consisting of related data items, such as name, address, phone, email-id, and so on. 2. Several types of storage media exist in computer systems. They are classified on the basis of speed with which they can access data. Among the media available are cache memory, main memory, flash memory, magnetic 201

disks, optical disks, and magnetic tapes. Database is typically stored on the magnetic disk because of its higher capacity and persistent storage. 3. The storage media that consist of high-speed devices are referred to as primary storage. The storage media that consist of slower devices are referred to as secondary storage. Magnetic disk (generally called disk) is the primary form of secondary storage that enables storage of enormous amount of data. The slowest media comes under the category of tertiary storage. Removable media, such as optical discs and magnetic tapes, are considered as tertiary storage. 4. The CPU does not directly access data on the secondary storage and tertiary storage. Instead, data to be accessed must move to main memory so that it can be accessed. 5. A major advancement in secondary storage technology is represented by the development of Redundant Arrays of Independent Disks (RAID). The idea behind RAID is to have a large array of small independent disks. Data striping is used to improve the performance of disk, which utilizes parallelism. Mirroring (also termed as shadowing) is a technique used to improve the reliability of the disk. In this technique, the data is redundantly stored on two physical disks. 6. Several different RAID organizations are possible, each with different cost, performance, and reliability characteristics. Raid level 0 to 6 exists. RAID level 1 (mirroring) and RAID level 5 are the most commonly used. 7. In Storage Area Network (SAN) architecture, many server computers are connected with a large number of disks on a high-speed network. Storage disks are placed at a central server room, and are monitored and maintained by system administrators. An alternative to SAN is Network-Attached Storage (NAS). NAS device is a server that allows file sharing. 8. The database is mapped into a number of different files, which are physically stored on disk for persistency. Each file is decomposed into equal size pages, which is the unit of exchange between the disk and main memory. 9. It is not possible to keep all the pages of a file in main memory at once; thus, the available space of main memory should be managed efficiently. 10. The main memory space available for storing the data is called buffer pool and the subsystem that manages the allocation of buffer space is called buffer manager. 11. Buffer manager uses replacement policy to choose a page for replacement if there is no space available for a new page in the main memory. The commonly used policies include LRU, MRU, and clock replacement policy. 12. There are two approaches to organize the database in the form of files. In the first approach, all the records of a file are of fixed-length. However, in the second approach, the records of a file vary in size. 13. Once the database is mapped to files, the records of these files can be mapped on to disk blocks. The two ways to organize them on disk blocks are spanned organization and unspanned organization. 14. There are various methods of organizing the records in a file while storing a file on disk. Some popular methods are heap file organization, sequential file organization, and hash file organization. 15. Heap file organization is the simplest file organization technique in which no particular order of record is maintained in the file. In sequential file, the records are physically placed in the sorted order based on the value of one or more fields, called the ordering field. The ordering field may or may not be the key field of the file, whereas records are organized using a technique called hashing in a hash file organization. 16. Hashing provides very fast access to an arbitrary record of a file, on the basis of certain search conditions. The hashing scheme in which a fixed number of buckets, say N, are allocated to a file to store records is called static hashing. Dynamic hashing technique allocates new buckets dynamically as needed. It allows

96%	MATCHING BLOCK 178/319	SA	248E1130_RDBMS.docx (D165247738)

the hash function to be modified dynamically to accommodate the growth or shrinkage of database

files. Two hashing techniques for files that grow and shrink in number of records dynamically—namely, extendible and linear hashing. 17. To speed up the retrieval of records, additional access structures called indexes are used. Index can be created or defined on any field termed as indexing field or indexing attribute of the file. Creating single-level indexes, multilevel indexes, and indexes on multiple keys are possible. 18. Different types of single-level index are primary index, clustering index, and secondary index. 19. A primary index is an ordered file that contains an index on the primary key of the file. An index defined on a sequentially ordered non-key attribute (containing duplicate values) of a file is called clustering index. The attribute on which the clustering index is defined is known as a clustering attribute. 202
20. A secondary index is an index defined on a non-ordered attribute of the file. The indexing attribute may be a candidate key or a non-key attribute of the file. 21. The type of index with multiple levels (two or more levels) of index is called multilevel index. Multilevel indexes can be implemented using B-trees and B + -trees, which are dynamic structures that allow an index to expand and shrink dynamically. B + -trees can generally hold more entries in their internal nodes than B-trees. 22. Partitioned hashing is an extension of static hashing, which allows access on multiple keys. It does not support range queries; however, it is suitable for equality comparisons. 203

CHAPTER 9 INTRODUCTION TO TRANSACTION PROCESSING After reading this chapter, the reader will understand: ? The four desirable properties of a transaction, which include atomicity, concurrency, isolation, and durability ? The various states that a transaction passes through during its execution ? Why the transactions are executed concurrently ? Anomalies due to interleaved execution of transactions ? The various possible schedules of executing transactions ? The concept of serializable schedules ? Different ways of defining schedule equivalence, which include result equivalence, conflict equivalence, and view equivalence ? Use of precedence graph in testing schedules for conflict serializability ? The recoverable and cascadeless schedules ? SQL transaction statements and transaction support in SQL

A collection of operations that form a single logical unit of work

is called a
transaction. The operations that make up a transaction typically consist of requests to access existing data, modify existing data, add new data, or any combination of these requests. The statements of a transaction are enclosed within the begin transaction and end transaction statements. For successful completion of a transaction and database changes to be permanent, the transaction must be completed in its entirety. That is, each step (or operation) in the transaction must succeed for the transaction to be successful. If any part of the transaction fails, then the entire transaction fails. An example of a transaction is paying of the utility bill from the customer's bank account which involves several operations such as debiting the bill amount from customer's account and crediting the amount to the utility provider's account. All the operations of fund transfer from customer's account to utility provider's account must succeed or fail as a group. It would be unacceptable if the customer's account is debited but the utility provider's account is not credited. This chapter first covers the basic concepts of transactions that include the desirable properties and various states of a transaction. It then discusses how concurrent execution of multiple transactions may lead to several anomalies. The chapter also discusses the various types of transaction schedules including serial, serializable, recoverable, and cascadeless schedules. The chapter concludes by giving an overview of transaction concept in SQL. 9.1 DESIRABLE PROPERTIES OF A TRANSACTION To ensure the integrity of the data, the database system must maintain some desirable properties of the transaction. These properties are known as ACID properties, the acronym derived from the first letter of the terms a tomicity, c onsistency, i solation, and d urability. Atomicity implies that either all of the operations that make up a transaction should execute or none of them should occur. It is the responsibility of the transaction management component of a DBMS to ensure atomicity. Consistency implies that if all the operations of a transaction are executed completely, the database is transformed from one consistent state to another. It is the responsibility of the application programmers who code the transactions to ensure consistency of the database. Note that during the execution of the transaction the state of the database becomes inconsistent. Such an inconsistent state of the database should not be visible to the users or other concurrently running transactions. Learn More If the operations in a transaction do not modify the database but only retrieve the data, the transaction is known as read- only transaction. Isolation implies that each transaction appears to run in isolation with other concurrently running transactions. That is, the execution of a transaction should not be interfered by any other concurrently running transaction. It is the responsibility of 243
the concurrency control component (discussed in Chapter 10) of the database system to allow concurrent execution of transactions without any interference from each other. Durability (also known as permanence) implies that once a transaction is completed successfully, the changes made by the transaction persist in the database, even if the system fails. The durability property is ensured by the recovery management component of the database system which is discussed in Chapter 11. Let us explain these ACID properties with the help of an example. Consider a bookshop that maintains the Online Book database to give details about the available books to its customers. The customers can order the books online and can make payments through credit card or debit card. Suppose that the customer pays the bill amount through debit card. The customer has to enter the debit card number and the pin number (secret number) for his or her verification. After the verification, the bill amount is immediately transferred from the customer's account to the shopkeeper's account. Consider the customer's account as account A and shopkeeper's account as account B. For the rest of the discussion, we will consider account A and account B instead of customer's account and shopkeeper's account, respectively. Consider a transaction T 1 that transfers $100 from account A to account B. Let the initial values of account A be $2000 and account B is $1500. The sum of the values of account A and B is $3500 before the execution of transaction T 1 . Since T 1 is a fund transferring transaction, the sum of the values of account A and B should be $3500 even after its execution. The transaction T 1 can be written as follows: T 1 :

89%	MATCHING BLOCK 176/319	W

read(A); A:=A–100; write(A); read(B); B:= B+100; write(B); If transaction

T 1 is executed, either $100 should be transferred from account A to B or neither of the accounts should be affected. If T 1 fails after debiting $100 from account A, but before crediting $100 to account B, the effects of this failed transaction on account A must be undone. This is the atomicity property of transaction T 1 .

55%	MATCHING BLOCK 177/319	W

The execution of transaction T 1 should also preserve the consistency of the database, that is, the sum of the values of account A and B should be $3500

even after the execution of T 1 . Now, let us consider the isolation property of T 1 . Suppose that during the execution of transaction T 1 when $100 is debited from account A and not yet credited to account B, another concurrently running transaction, say T 2 reads the values of account A and B. Since T 1 has not yet completed, T 2 will read inconsistent values. The isolation property ensures that the effects of the transaction T 1 are not visible to other transaction T 2 until T 1 is completed. If the transaction T 1 is successfully completed and the user who initiated the transaction T 1 has been notified about successful transfer of funds, then the data regarding the transfer of funds should not be lost even if the system fails. This is the durability property of T 1 . The durability can be guaranteed

43%	MATCHING BLOCK 180/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

by ensuring that either ? all the changes made by the transaction are written to the disk before the transaction completes, or ? the information about all the changes made by the transaction is written to the disk and is sufficient to enable the database system to reconstruct the changes when the database system is restarted after the failure. 9.2

STATES OF A TRANSACTION

100%	MATCHING BLOCK 179/319	W

Whenever a transaction is submitted to a DBMS for execution,

either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are active, partially committed, committed, failed, and aborted. Figure 9.1 shows the state transition diagram that describes how a transaction passes through various states during its execution. 244
Fig. 9.1 State transition diagram showing various states of a transaction A transaction enters into the active state with its commencement. At this point, the system marks BEGIN_TRANSACTION operation to specify the beginning of the transaction execution. During its execution, the transaction stays in the active state and executes several READ and WRITE operations on the database. The READ operation transfers a data item from the database to a local buffer of the transaction that has

77%	MATCHING BLOCK 181/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

executed the read operation. The WRITE operation transfers the data item from the local buffer of the transaction back to the database.

Once the transaction executes its final operation, the system marks END_TRANSACTION operation to specify the end of the transaction execution. At this point, the transaction enters into partially committed state. The actual output at this point may still be residing in the main memory and thus, any kind of hardware failure might prevent its successful completion. In such a case, the transaction may have to be aborted. Before actually updating the database on the disk, the system first writes the details of updates performed by the transaction in the log file (discussed in Chapter 11). The log file is then written to the disk so that, even in case of failure, the system can re-construct the updates performed by the transaction when the system restarts after the failure. When this information is successfully written out in the log file, the system marks COMMIT_TRANSACTION operation to indicate the successful end of the transaction. Now, the transaction is said to be committed and all its changes must be reflected permanently in the database. If the transaction is aborted during its active state or the system fails to write the changes in log file, the transaction enters into the failed state. The failed transaction must be rolled back to undo its effects on the database to maintain the consistency of the database. When the transaction leaves the system, it enters into the terminated state. At this point, the transaction information maintained in the log file during its execution is removed. An aborted transaction can be restarted either automatically or by the user. A transaction can only be restarted automatically by the system if it was aborted due to some hardware or software error and not due to some internal logical error of the transaction. If the transaction was aborted due to some internal logic error or because the desired data was not found in the database, then it must be killed by the database system. In this case, the user must first rectify the error in the transaction and then submit it again to the system as a new transaction for execution. Learn More There are some situations when a transaction needs to perform write operations on a terminal or a printer. These writes are known as observable external writes. Since the content once written on the printer or the terminal cannot be erased, the database system allows such writes to take place only after the transaction has entered the committed state. 245
Note that once a transaction has been committed, its effects cannot be undone by aborting it.

97%	MATCHING BLOCK 182/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

The only way to undo the effects of a committed transaction is to execute a compensating transaction,

which reverses the effect of a committed transaction. For example, a transaction that has added $100 in an account can be reversed back by executing a compensating transaction that would subtract $100 from the account. However, it is not always possible to create a compensating transaction. Therefore, it is the responsibility of the user, and not of the database system, to write and execute a compensating transaction. 9.3 CONCURRENT EXECUTION OF TRANSACTIONS The transaction-processing system allows concurrent execution of multiple transactions to improve the system performance. In concurrent execution, the database management system controls the execution of two or more transactions in parallel; however, allows only one operation of any transaction to occur at any given time within the system. This is also known as interleaved execution of multiple transactions. The database system allows concurrent execution of transactions due to two reasons. First, a transaction performing read or write operation using I/O devices may not be using the CPU at a particular point of time. Thus, while one transaction is performing I/O operations, the CPU can process another transaction. This is possible because CPU and I/O system in the computer system are capable of operating in parallel. This overlapping of I/O and CPU activities reduces the amount of time for which the disks and processors are idle and, thus,

94%	MATCHING BLOCK 183/319	SA	DCAP 011.docx (D142453284)

increases the throughput of the system (the number of transactions executed in a given amount of time).

Second, interleaved execution of a short and a long transaction allows short transaction to complete quickly. If a short transaction
is executed after the execution of a long transaction in a serial order, the

57%	MATCHING BLOCK 185/319	SA	DCAP 011.docx (D142453284)

short transaction may have to wait for a long time, leading to unpredictable delays in executing a transaction. On the

other hand, if these transactions are executed concurrently, it reduces the unpredictable delays, and thereby reduces the average response time. Note that despite the correctness of each individual transaction, undesirable interleaving of multiple transactions may lead to database inconsistency. Thus, interleaving should be done carefully to ensure that the result of a concurrent execution of transactions is equivalent to some serial execution (one after the other) of the same set of transactions. The database system must control the interaction among concurrently executing transactions. The database system does so with the help of several concurrency-control techniques which are discussed in Chapter 10. 9.3.1 Anomalies Due to Interleaved Execution If the transactions are interleaved in an undesirable manner, it may lead to several anomalies such as lost update, dirty read, and unrepeatable read. To understand the effect of these anomalies on the database, consider two transactions T 1 and T 2 , where T 1 transfers $100 from account A to account B, and T 2 adds two percent interest to account A. Suppose that the initial values of account A and B are $2000 and $1500, respectively. Then, after the serial execution of transactions T 1 and T 2 (T 1 followed by T 2), the value of account A should be $1938 and that of account B should be $1600. Suppose that the operations of T 1 and T 2 are interleaved in such a way that T 2 reads the value of account A before T 1 updates its value in the database. Now, when T 2 updates the value of account A in the database, the value of account A updated by the transaction T 1 is overwritten and hence, is lost. This is known as lost update problem. The value of account A at the end of both the transactions is $2040 instead of $1938 which leads to data inconsistency. The interleaved execution of transactions T 1 and T 2 that leads to lost update problem is shown in Figure 9.2. 246
Fig. 9.2 Lost update problem The second problem occurs when a transaction fails after updating a data item, and before this data item is changed back to its original value, another transaction reads this updated value. For example, assume that T 1 fails after debiting $100 from account A, but before crediting this amount to account B. This will leave the database in an inconsistent state. The value of account A is now $1900, which must be changed back to original one, that is, $2000. However, before the transaction T 1 is rolled back, let another transaction T 2 reads the incorrect value of account A. This incorrect value of account A that is read by transaction T 2 is called dirty data, and the problem is called dirty read problem. The interleaved execution of transactions T 1 and T 2 that leads to dirty read problem is shown in Figure 9.3. Fig. 9.3 Dirty read problem The third problem occurs when a transaction tries to read the value of the data item twice, and another transaction updates the same data item in between the two read operations of the first transaction. As a result, the first transaction reads varied values of same data item during its execution. This is known as unrepeatable read. For example, consider a transaction T 3 that reads the value of account A. At this point, let another transaction T 4 updates the value of account A. Now if T 3 again tries to read the value of account A, it will get a different value. As a result, the transaction T 3 receives different values for two reads of account A. The interleaved schedule of transactions T 3 and T 4 that leads to a problem of unrepeatable read is shown in Figure 9.4. Learn More The three anomalies, namely, lost update, dirty read, and unrepeatable read can also be described in terms of when the actions of two transactions, say T 1 and T 2 , conflict with each other. In this case, the lost update problem is known as write- write (WW) conflict, the dirty read problem is known as write-read (WR) conflict, and unrepeatable problem is known as read-write (RW) conflict. 247
Fig. 9.4 Unrepeatable read 9.4 TRANSACTION SCHEDULES A list of operations (such as reading, writing, aborting or committing) from a set of transactions is known as a schedule (or history). A schedule should comprise all the instructions of the participating

90% **MATCHING BLOCK 184/319** W

transactions and also preserve the order in which the instructions appear in each individual transaction.

54% **MATCHING BLOCK 186/319** SA BOOK SIZE.docx (D50092775)

Consider two transactions T 1 and T 5 , where T 1 transfers $100 from account A to account B and T 5 transfers $50 from account B to account

C. Here, account C is another account of the shopkeeper. Suppose that the values of account A, B, and C are $2000, $1500, and $500, respectively. The sum A + B + C is $4000. Also suppose that the two transactions are executed one after the other in the order T 1 followed by T 5 . Figure 9.5 shows the serial execution of transactions T 1 and T 5 . This sequence of execution of the transactions is called a serial schedule. A serial schedule consists of a sequence of instructions from various transactions, where the operations of one single transaction appear together in that schedule. Hence, for a set of n transactions, n! different valid serial schedules are possible. Fig. 9.5 Schedule 1—A serial schedule in the order T 1 followed by T 5 In Figure 9.5, the sequence of instructions is in chronological order from top to bottom with the instructions of transaction T 1 appearing in the left column followed by the instructions of T 5 appearing in the right column. The final values of accounts A, B, and C after the execution of both the transactions are $1900, $1550, and $550, respectively. The sum A + B + C is preserved after the execution of both the transactions. Similarly, if the transaction T 1 is executed after the execution of transaction T 5 , then also the sum A + B + C is preserved. The serial schedule of transactions T 1 and T 5 in the order T 5 followed by T 1 is shown in Figure 9.6. 248
Fig. 9.6 Schedule 2—A serial schedule in the order T 5 followed by T 1 A schedule can also be described with the help of shorthand notation that uses the symbols

r, w, c, and a, for the operations read, write, commit, and abort, respectively.

Note that

for the purpose of recovery and concurrency control, we are only interested

in these four operations. The transaction id (transaction number) is also appended as a subscript to each operation in the schedule. For example, Schedule 1 of Figure 9.5, which may be named S 1 , can be expressed as given here. S 1 :

r 1 (A); w 1 (A); r 1 (B); w 1 (B); r 5 (B); w 5 (B); r 5 (C); w 5 (C);

Each transaction in a serial schedule is executed independently without any interference from the operations of other transactions. As long as every transaction is executed from beginning to end without any interference from other transactions, it gives a correct end result on the database. Therefore, every serial schedule is considered to be correct. However, in serial schedules only one transaction is active at a time. The commit or abort operations of the active transaction initiates the next transaction. The transactions are not interleaved in a serial schedule. Thus, if the active transaction is waiting for an I/O operation, CPU cannot be used for another transaction, which results in CPU idle time. Moreover, if one transaction is quite long, the other transactions have to wait for long time for their turn. Hence, serial schedules are unacceptable. 9.4.1 Serializable Schedules In a multi-user database system, several transactions are executed concurrently for efficient use of system resources. When two transactions are executed concurrently, the operating system may execute one transaction for some time, then perform a context switch and execute the second transaction for some time, and then switch back to the first transaction, and so on. Thus, when multiple transactions are executed concurrently, the CPU time is shared among all the transactions. The schedule resulted from this type of interleaving of operations from various transactions is known as non-serial schedule. Learn More If the operating system is given the entire responsibility of executing the transactions concurrently, then it can even generate the schedules that can leave the database in inconsistent state. Therefore, it is the responsibility of concurrency- control component of database system to ensure that only those schedules should be executed that will leave the database in a consistent state. In general, there is no way to predict exactly how many operations of a transaction will be executed before the CPU switches to another transaction. Thus, for a set of n transactions, the number of possible non-serial schedules is much larger than n!. However, not all non-serial schedules result in the correct state of the database. For example, the schedule for the transactions T 1 and T 5 shown in Figure 9.7 is not a correct schedule as it leaves the database in an inconsistent 249

state. After the execution of this schedule, the final values of accounts A, B, and C are $1900, $1600, and $550, respectively. Thus, the sum A + B + C is not preserved. Fig. 9.7 Schedule 3—A concurrent schedule resulting in an inconsistent state of database The consistency of the database under concurrent execution can be ensured by interleaving the operations of transactions in such a way that the final output is same as that of some serial schedule of those transactions. Such a schedule is referred to as serializable schedule. Thus,
a schedule S of n transactions T 1 ,

T 2 , T 3 , ..., T n is serializable if it is equivalent to some serial schedule of the same n transactions.

Figure 9.8 shows a non-serial schedule of transactions T 1 and T 5 which is equivalent to Schedule 1. After the execution of this schedule, the final values of accounts A, B, and C are $1900, $1550, and $550. Thus, the sum A + B + C is preserved and hence, it is a serializable schedule. The shorthand notation for this schedule is given here. S 4 :

r 1 (A); w 1 (A); r 5 (B); w 5 (B); r 1 (B); w 1 (B); r 5 (C); w 5 (C);

Fig. 9.8 Schedule 4—A concurrent schedule resulting in consistent state of the database 9.4.2 Schedules Equivalence Two different schedules may produce the same final state of the database. Such schedules are known as result equivalent, since they have same effect on the database. However, in some cases, two different schedules may accidentally produce the same final state of database. For example, consider two schedules S i and S j that are updating the data item Q, as shown in the Figure 9.9. 250
Fig. 9.9 Schedules S i and S j Suppose that value of data item Q is $100 then the final state of database produced by schedules S i and S j is same, that is, they produce the same value of Q ($200). However, if the initial value of data item Q is other than $100, the final state may not be the same. For example, if the initial value of Q is $200, then these schedules will not produce the same final state. Thus, with this initial value, the schedules S i and S j are not result equivalent. Moreover, these two schedules execute two different transactions and hence, cannot be considered as equivalent. Thus, result equivalence alone is not the sufficient condition for defining the equivalence of the schedules. Two other most commonly used forms of schedule equivalence are conflict equivalence and view equivalence that lead to the notions of conflict serializability and view serializability. Conflict Equivalence and Conflict Serializability

Two operations in a schedule are said to conflict if they belong to different transactions, access the same data item, and at least one of

them is
the write operation. On the other hand, two operations belonging to different transactions in a schedule do not conflict if both of them are read operations or both of them are accessing different data items. To understand the concept of conflicting operations in a schedule, consider two transactions T 6 and T 7 that are updating the data items Q and R in the database. A non-serial schedule of these transactions is shown in Figure 9.10. Here, we have taken only read and write operations as these are the only significant operations from the scheduling point of view. Fig. 9.10 Schedule 5—showing only read and write operations In Schedule 5, the write(Q) operation of T 6 conflicts with the read(Q) operation of T 7 because both the operations are accessing the same data item Q, and one of these operations is the write operation. On the other hand, the write(Q) operation of T 7 is not conflicting with read(R) operation of T 6 , because both are accessing different data items Q and R. The order of execution of two conflicting operations is important because if they are applied in different order, they can have different

effect on the database or on the other transactions in the

schedule. For example, if write(Q) operation of T 6 is swapped with read(Q) operation of T 7 , the value read by the read(Q) operation can be different. However, the order of execution of non-conflicting operations does not matter. It means that if write(Q) operation of T 7 is swapped with read(Q) operation of T 6 , then a new schedule is produced, which will have the same effect on the database as Schedule 5. The equivalent schedule (Schedule 6) in which all the operations appear in the same order as that of the Schedule 5 except for the write(Q) and read(R) operations of T 7 and T 6 , respectively is shown in Figure 9.11. Note that the schedules shown in Figures 9.10 and 9.11 produce the same final state of the database regardless of initial system state. 251
Fig. 9.11 Schedule 6 — after swapping nonconflicting operations of Schedule 5 Similarly, we can reorder the following non-conflicting operations in the Schedule 5, without affecting its impact on the database. ? read(R) operation of T 6 is swapped with read(Q) operation of T 7 ? write(R) operation of T 6 is swapped with write(Q) operation of T 7 ? write(R) operation of T 6 is swapped with read(Q) operation of T 7 The resultant schedule generated from the sequence of these steps is shown in Figure 9.12. Note that this schedule is a serial schedule. Thus, we have shown that Schedule 5 is equivalent to a serial schedule (Schedule 7). This equivalence means that regardless of the initial system state, Schedule 5 will produce the same final state as some serial schedule.
Fig. 9.12 Schedule 7—serial schedule equivalent to Schedule 5

85%	MATCHING BLOCK 195/319	SA	DBMS Block 2.pdf (D149011992)

If a schedule S can be transformed into a schedule S′ by performing a series of swaps of non-conflicting operations, then S and S′ are conflict equivalent.

In other words, two schedules are said to be

96%	MATCHING BLOCK 194/319	W	

conflict equivalent if the order of any two conflicting operations is same in both

the schedules. For example, Schedule 5 and Schedule 6

66%	MATCHING BLOCK 196/319	SA	BOOK SIZE.docx (D50092775)

are conflict equivalent. The concept of conflict equivalence leads to the concept of conflict serializability. A schedule, which is conflict equivalent to some serial schedule,

is known as conflict serializable. Hence, Schedule 5 is a conflict serializable schedule as it is conflict equivalent to a serial schedule (Schedule 7). To better understand the concept of conflict equivalent and conflict serializable schedules, consider Schedule 8 and Schedule 9 of transactions T 8 and T 9 shown in Figure 9.13. The write(Q) operation of T 8 and read(Q) operation of T 9 are

61%	MATCHING BLOCK 197/319	SA	Database System.pdf (D165747767)

conflicting operations in both the schedules and the order of these operations in both the schedules is

also same. Thus, Schedule 8 and Schedule 9 are conflict equivalent. Moreover, Schedule 9 is a serial schedule; hence, Schedule 8 is conflict serializable schedule. 252
Fig. 9.13 Conflict serializable schedules Testing Schedules for Conflict Serializability The conflict serializability of a schedule can be tested using a simple algorithm that considers only read and write operations in a schedule to construct a

90%	MATCHING BLOCK 200/319	W	

precedence graph (or serialization graph). A precedence graph is a directed graph G = (N,E) where N = {T 1 , T 2 , ..., T n } is a set of nodes and E = {e 1 , e 2 , ..., e n } is a set of directed edges. For each transaction T i in a schedule, there exists a node in the graph. An edge e i in the graph is shown as (T i → T j), where T i is the starting node and T j is the ending node of the edge e i . The edge e i is created if one of the operations in T j appears in the schedule before some conflicting operation in T j . The algorithm to test the conflict serializability of a schedule S is given here. 1. For each transaction T i

participating in the schedule S, create a node labelled T

i . 2. If T j executes a read operation after T i executes a write operation on any data item say Q,

create an edge (T i → T j) in the precedence graph. 3. If T j executes a write operation after T i executes a read

operation on any data item say Q,

create an edge (T i → T j) in the precedence graph. 4. If T j executes a write operation after T i executes a write

operation on any data item say Q,

create an edge (T i → T j) in the precedence graph. 5.

If the resultant precedence graph is acyclic, that is, it does not contain any cycle, then the schedule S is considered to be conflict serializable. If a

precedence graph contains a cycle, the schedule S is not conflict serializable.

Some examples of cyclic and acyclic precedence graphs are shown in Figure 9.14. Note that the edges (T i → T j) in a precedence graph can also be labelled by the name of the data item that led to the creation of the edge (see Figure 9.15). Fig. 9.14 Acyclic and cyclic precedence graphs An edge from T i to T j in the precedence graph implies that the transaction T i must appear before T j

in any serial schedule S′ that is equivalent to S.

Thus, if there are no cycles in the precedence graph, a serial schedule S′ equivalent to S can be created by ordering the transactions in such a way that whenever

an edge (T i → T j) exists in the precedence graph,

T i must appear before T j in the equivalent serial schedule S′. The process of ordering the nodes of an acyclic preceding graph is known as topological sorting. The topological ordering produces equivalent serial schedule. For example, consider Schedule 5 shown in Figure 9.10. The precedence graph of this schedule is shown in Figure 9.15(a). The precedence graph contains two nodes, one for each transaction—T 6 and T 7 . An edge from T 6 to T 7 is created which is labelled with the data items Q and R that led to the creation of this edge. Since there are no cycles in the precedence graph, Schedule 5 is conflict serializable schedule. The serial schedule equivalent to Schedule 5 is shown in Figure 9.15(b). 253
Fig. 9.15 An example of testing conflict serializability Though only one serial schedule exists for Schedule 5, in general, there can be more than one equivalent serial schedule for a conflict serializable schedule. For example, consider a precedence graph for a schedule of three transactions T 1 , T 2 , and T 3 shown in Figure 9.16(a). It has two equivalent serial schedules shown in Figure 9.16(b). Fig. 9.16 Precedence graph with two equivalent schedules To better understand the concept of testing a conflict serializability of a schedule, consider Schedule 10 of three transactions T 10 , T 11 , and T 12 shown in Figure 9.17(a). In this schedule, the read(Q) operation of T 10 and write(Q) operation of T 12 are the conflicting operations. Similarly, the write(Q) operation of T 12 and read(Q) operation of T 11 are the conflicting operations. The write(R) operation of T 11 and read(R) operation of T 10 are also conflicting operations. Thus, a precedence graph with edges from T 10 to T 12 , T 12 to T 11 , and T 11 to T 10 is created [see Figure 9.17(b)]. Since this graph contains cycle (T 10 → T 12 → T 11 → T 10), the Schedule 10 is not conflict serializable. Therefore, no serial schedule equivalent to this schedule will exist. Fig. 9.17 Schedule 10 and its precedence graph View Equivalence and View Serializability Equivalence of schedules can also be determined by another less restrictive definition of equivalence called
view equivalence.
Two schedules
S and S′

74%	MATCHING BLOCK 205/319	W

are said to be view equivalent if the schedules satisfy these conditions. 1. The same set of transactions participates in S and S′, and S and S′ include the same set of

operations of those transactions. 2. If the transaction T i reads the initial value of a data item say Q

90%	MATCHING BLOCK 206/319	SA	Advanced DBMS.pdf (D166063498)

in schedule S, then T i must read the initial value of

same data item Q in schedule S′ also. 3. For each data item Q, if T i executes read(Q) operation after the write(Q) operation of transaction T j in schedule S, then T i must execute the read(Q) operation after the write(Q) operation of T j in schedule S′ also. 254
4. If the transaction T i performs the final write operation on any data item say Q in schedule S, then it must perform final write operation on the same data item Q in schedule S′ also. Conditions 2 and 3 ensure that as long as each transaction reads the same values in both the schedules, they perform the same computation. Conditions 4 (along with conditions 2 and 3) ensure that the final write operation on each data item is same in both the schedules and hence, produces the same final state of the database.

75%	MATCHING BLOCK 208/319	SA	BOOK SIZE.docx (D50092775)

The concept of view equivalence leads to the concept of view serializability. A schedule S is said to be view serializable if it is view equivalent to some serial schedule.

Fig. 9.18 View serializable schedules To better understand the concept of view equivalence and view serializability, consider Schedule 11 and Schedule 12 of three transactions T 13 , T 14 , and T 15 shown in Figure 9.18. In both these schedules, the transaction T 14 reads the initial value of data item Q (condition 2). Moreover, in both the schedules, the transaction T 13 performs the read(R) operation after the write(R) operation of T 14 (condition 3). Similarly, the transaction T 15 performs read(R) operation after the write(R) operation of T 14 in both the schedules. Finally, in both the schedules, the transaction T 15 performs the final write(Q) operation (condition 4). Thus, Schedule 11 and Schedule 12 are view equivalent. Moreover, Schedule 12 is a serial schedule hence, Schedule 11 is view serializable also. Now, consider Schedule 13 given in Figure 9.19. In this schedule, the transactions T 17 and T 18 perform write(Q) operation without performing the read(Q) operation. Thus, the value written by the write(Q) operation is independent of the old value of Q in the database. These types of write operations are known as blind writes. This schedule is view equivalent to some serial schedule of transactions T 16 , T 17 , and T 18 . Hence, it is view serializable. However, it is not conflict serializable since its precedence graph is cyclic (see Figure 9.20). Thus,

88%	MATCHING BLOCK 209/319	SA	BOOK SIZE.docx (D50092775)

blind writes appear in a view serializable schedule that is not conflict serializable.

Learn More The definitions of conflict serializability and view serializability are similar if a condition known as the constrained write assumption holds on all transactions in a given schedule. It states that if any write operation performed on a data item P in 255
transaction T i is preceded by a read operation on the same data item P in T i , the value of P written by the write operation in T i depends only on the value of P read by that read(P) operation. Every conflict serializable schedule is also view serializable but not vice-versa. Fig. 9.19 Schedule 13—showing an example of blind write Fig. 9.20 Precedence graph of Schedule 13 9.4.3 Recoverable and Cascadeless Schedules The computer systems are prone to failures. Hence, it is important to consider the effects of transaction failure during concurrent execution. As discussed earlier, if a transaction T i fails due to any reason; its effects need to be undone to ensure atomicity property of the transaction. During concurrent execution of transactions, it must be ensured that any transaction T j that depends on T i (that is,

85%	MATCHING BLOCK 210/319	SA	DBMS Block 2.pdf (D149011992)

T j has read the value of data item written by T i) must also be rolled back.

Thus, some restrictions must be placed on the type of schedules permitted in the system. Two types of schedules, namely, recoverable and cascadeless schedules are acceptable from the viewpoint of recovery from transaction failure. If, in a

78%	MATCHING BLOCK 211/319	SA	U234.pdf (D109498494)

schedule S, a transaction T j reads a data item written by T i , then the commit operation of T i should appear before the commit operation of T j .

Such a schedule is known as recoverable schedule. Consider Schedule 14 of transactions T 19 and T 20 as shown in the Figure 9.21. The transaction T 20 performs read(Q) operation after the write(Q) operation of T 19 . Suppose that immediately after read(Q) operation, transaction T 20 commits. Now if transaction T 19 fails, its effects need to be undone in order to ensure atomicity of the transaction. Since, T 20 has read the value of Q written by T 19 , it also has to be rolled back. However, T 20 has already been committed, thus, cannot be rolled back. In such situations, it becomes impossible to recover from the failure of transaction T 19 . Such schedules are known as non-recoverable schedule s. The database system must ensure that the schedules are recoverable. If the commit operation of transaction T 20 in Figure 9.21 is performed after the commit operation of T 19 , the schedule becomes recoverable. Fig. 9.21 Schedule 14 256
Sometimes, even if a schedule is recoverable, several transactions may have to be rolled back in order to recover completely from the failure of a transaction T i . It happens if transactions have read the value of a data item written by T i . For example, consider Schedule 15 shown in Figure 9.22. In this schedule,

87%	MATCHING BLOCK 212/319	SA	BOOK SIZE.docx (D50092775)

the transaction T 22 reads the value of data item Q written by

transaction T 21 , and transaction T 23 reads the value of data item Q written by transaction T 22 . Fig. 9.22 Schedule 15 Now suppose that the transaction T 21 fails due to any reason, it has to be rolled back. Since transaction T 22 has read the data written by T 21 , it also needs to be rolled back. The transaction T 23 also needs to be rolled back because it has in turn read the data written by T 22 . Such a situation, in which failure of

76%	MATCHING BLOCK 213/319	SA	DBMS Block 2.pdf (D149011992)

a single transaction leads to a series of transaction rollbacks, is called cascading rollback. Cascading rollback requires undoing of significant amount of work

and thus, is undesirable. Therefore, schedules should be restricted to avoid cascading rollbacks. The schedules that avoid cascading rollbacks are known as cascadeless schedule. In cascadeless schedules, transactions T i and T j are interleaved in such a way

95%	MATCHING BLOCK 214/319	SA	U234.pdf (D109498494)

that T j reads a data item previously written by T i ; however, commit operation of T i appears before the read operation of T j . Every cascadeless schedule is also recoverable. 9.5

SQL TRANSACTION STATEMENTS As discussed earlier that the statements of a transaction are enclosed within the begin transaction and end transaction statements. However, in SQL:92 standard, there is no explicit BEGIN TRANSACTION statement. A transaction initiates implicitly when SQL statements are encountered. However, an explicit end statement is required which can be COMMIT or ROLLBACK statement. Execution of a COMMIT statement or ROLLBACK statement completes the current transaction. The COMMIT statement terminates the current transaction and makes all the changes made by the transaction permanent in the database. That is, it commits the changes to the database. The COMMIT statement has the following general format. COMMIT [WORK] where, WORK is an optional keyword that does not change the semantics of COMMIT The ROLLBACK statement, on the other hand, terminates the current transaction and undoes all the changes made by the transaction. That is, it rolls back the changes to the database. The ROLLBACK statement has the following general format. ROLLBACK [WORK] In this case also, WORK is an optional keyword that does not change the semantics of ROLLBACK. 9.5.1 Transaction Characteristics in SQL Every transaction in SQL has certain characteristics associated with it such as access mode, diagnostic area size, and isolation level. The SET TRANSACTION statement in SQL:92 is used to specify these characteristics.

100%	MATCHING BLOCK 215/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

The access mode can be specified as READ ONLY or READ WRITE. The

READ ONLY mode allows only retrieval of data from the database. The READ WRITE mode, on the other hand,

100%	MATCHING BLOCK 217/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

allows UPDATE, INSERT, DELETE, and CREATE commands to be executed. 257

The diagnostic area size option, specified as DIAGNOSTIC SIZE n, determines the number of error or exception conditions that can be recorded in the diagnostic area for the current transaction. The diagnostic area is populated for each SQL statement that is executed. If the statement raised more conditions than the specified DIAGNOSTIC SIZE, only most severe conditions will be reported. For example, if we specify DIAGNOSTIC SIZE 5 but one of the SQL statements raised 10 conditions, it is most likely that only "most severe" 5 conditions will be reported. SQL:92 also supports four levels of transaction isolation. The isolation levels reduce the isolation between concurrently executing transactions. The level providing the greatest isolation from other transactions is SERIALIZABLE. It is the default level of isolation. At this level, a transaction is fully isolated from the changes made by other transactions. The other isolation levels are READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ. READ UNCOMMITTED is the lowest level of isolation. At this level, a transaction can read subsequent changes made by other transactions, either committed or uncommitted. With read uncommitted isolation level, one or more of the three problems, namely, dirty read, unrepeatable read, and phantom read may occur. We have already discussed the dirty read and unrepeatable read problems. Phantom read problem occurs when

76%	MATCHING BLOCK 216/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

a transaction T i reads a set of rows from a table based on some condition specified in WHERE clause,

meanwhile another transaction T j inserts a new row into that table that also satisfies the condition and commits. Now, if T i is again executed, then it will see a tuple (or phantom tuple) that previously did not exist. The next level above READ UNCOMMITTED is READ COMMITTED, and above that is REPEATABLE READ. In READ COMMITTED isolation level, dirty reads are not possible since a transaction is not allowed to read the updates made by uncommitted transactions. However, unrepeatable reads and phantoms are possible. In REPEATABLE READ isolation level, dirty reads and unrepeatable reads are not possible but phantoms are possible. In SERIALIZABLE isolation level, dirty reads, unrepeatable reads, and phantoms are not possible. The possible violations for different transaction isolation levels are summarized in Table 9.1. Table 9.1 Possible violations for different isolation levels An example of an SQL transaction consisting of multiple SQL statements is given in Figure 9.23. In this transaction, two SQL statements are given that update the BOOK and PUBLISHER relations of Online Book database. The first statement updates the price of all the books published by publisher P001. The next statement updates the email-id of the publisher with P_ID="P002".

59%	MATCHING BLOCK 221/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

If an error occurs on any SQL statement, the entire transaction is rolled back. That is, any updated value would be restored to its original value.

EXEC SQL WHENEVER SQLERROR GOTO mylabel; EXEC SQL SET TRANSACTION READ WRITE, ISOLATION LEVEL SERIALIZABLE, DIAGNOSTIC SIZE 10; EXEC SQL UPDATE BOOK SET Price=Price*1.2 WHERE P_ID="P001"; EXEC SQL UPDATE PUBLISHER SET Email_id="sunsh@sunshinepub.com" 258
WHERE P_ID="P002"; EXEC SQL COMMIT; GOTO LAST; mylabel: EXEC SQL ROLLBACK; LAST:...; Fig. 9.23 An example of SQL transaction SUMMARY 1.

100%	MATCHING BLOCK 218/319	SA	Advanced DBMS.pdf (D166063498)

A collection of operations that form a single logical unit of work

is called a
transaction. The statements of a transaction are enclosed within the begin transaction and end transaction statements. 2. For a transaction to be completed and database changes to be made permanent, a transaction has to be completed in its entirety. 3. To ensure the integrity of the data, the database system must maintain some desirable properties of the transaction. These properties are known as ACID properties, the acronym derived from the first letter of the terms atomicity, consistency, isolation, and durability. 4. Atomicity implies that either all of the operations that make up a transaction should execute or none of them should occur. 5. Consistency implies that if all the operations of a transaction are executed completely without the interference of other transactions, the consistency of the database is preserved. 6. Isolation implies that each transaction appears to run in isolation with other concurrently running transactions. 7. Durability (also known as permanence) implies that once a transaction is completed successfully, the changes made by the transaction persist, even if the system fails. 8.

100%	MATCHING BLOCK 219/319	W
Whenever a transaction is submitted to a DBMS for execution,		

either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states such as active, partially completed, committed, failed, and aborted. 9. The transaction-processing system allows concurrent execution of multiple transactions to improve the system performance. 10. In concurrent execution, the database management system controls the execution of two or more transactions in parallel; however, allows only one operation of any transaction to occur at any given time within the system. This is also known as interleaved execution of multiple transactions. 11. The undesirable interleaved execution of transactions may lead to certain problems such as lost update, dirty read, and unrepeatable read. 12. A list of operations (such as reading, writing, aborting, or committing) from a set of transaction is known as a schedule (or history). A schedule should comprise all the instructions of the participating

90%	MATCHING BLOCK 220/319	W
transactions and also preserve the order in which the instructions appear in each individual transaction. 13. A		

serial schedule consists of a sequence of instructions from various transactions, where the operations of one single transaction appear together in that schedule. 14. The consistency of the database under concurrent execution can be ensured by interleaving the operations of transactions in such a way that the final output is same as that of some serial schedule of those transactions. Such a schedule is referred to as serializable schedule. 15. There are several ways to define schedule equivalence. The simplest way of defining schedule equivalence is to compare the result of the schedules on the database. Two schedules are known as result equivalent if they produce same final state of the database. 16. Two other most commonly used forms of schedule equivalence are conflict equivalence and view equivalence that lead to the notions of conflict serializability and view serializability. 259
17.
Two operations in a schedule are said to conflict if they belong to
different

79%	MATCHING BLOCK 223/319	SA	BOOK SIZE.docx (D50092775)
transactions, access the same data item and at least one of them is write operation. 18. Two schedules are said to be conflict equivalent if			

the order of any two conflicting operations is same in both
the schedules. A schedule, which is conflict equivalent to some serial schedule,
is known as conflict serializable. 19. The conflict serializability of a schedule can be tested using a simple algorithm that considers only read and write operations in a schedule to construct a precedence
graph.

86%	MATCHING BLOCK 222/319	SA	Advanced DBMS.pdf (D166063498)
A precedence graph is a directed graph consists of a set of nodes and a set of directed edges. 20.			

76%	MATCHING BLOCK 227/319	SA	DBMS.docx (D142535242)
If the precedence graph is acyclic then the schedule is considered to be conflict serializable.			

If a

100%	MATCHING BLOCK 224/319	SA	Advanced DBMS.pdf (D166063498)
precedence graph contains a cycle, the schedule is not conflict serializable. 21.			

The process of ordering the nodes of an acyclic preceding graph is known as topological sorting. 22. Equivalence of schedules can also be determined by another less restrictive definition of equivalence called view equivalence.

91%	MATCHING BLOCK 225/319	W
A schedule S is said to be view serializable if it is view equivalent to some serial schedule. 23.		

The
computer systems are prone to failure. Hence, it is important to consider the effects of transaction failure during concurrent execution. 24. A situation, in which failure of

96%	MATCHING BLOCK 226/319	W
single transaction leads to a series of transaction rollbacks, is called cascading rollback.		

The database system must ensure that the schedules are recoverable and cascadeless. 260
CHAPTER 10 CONCURRENCY CONTROL TECHNIQUES
After reading this chapter, the reader will understand: ? The need of concurrency control techniques ? The basic concept of
locking, types of locks and their implementation ? Lock based techniques for concurrency control ? Two-phase locking and its different variants ? Specialized locking techniques, such as multiple-granularity locking, tree structured indexes, etc. ? Factors affecting the performance of locking ? Timestamp-based techniques for concurrency control ? Variants of timestamp ordering techniques ? Optimistic techniques for concurrency control ? Multiversion technique based on timestamp ordering ? Multiversion technique based on locking ? Handling deadlock ? Different deadlock prevention techniques ? Deadlock detection and recovery Transactions execute either serially or concurrently. If all the transactions are restricted to execute serially, isolation property of transaction is maintained and the database remains in a consistent state. However, executing transactions serially unnecessarily reduce the resource utilization. On the other hand, execution of several transactions concurrently may result in interleaved operations and isolation property of transaction may no longer be preserved. Thus, it may leave the database in an inconsistent state. Consider a situation in which two transactions concurrently access the same data item. One transaction modifies a tuple, and another transaction makes a decision on the basis of that modification. Now, suppose that the first transaction rolls back. At this point, the decision of second transaction becomes invalid. Thus, concurrency control techniques are required to control the interaction among concurrent transactions. These techniques ensure that the concurrent transactions maintain the integrity of a database by avoiding the interference among them. This chapter discusses the concurrency control techniques, which ensure serializability order in the schedule. These techniques are locking, timestamp-based, optimistic, and the multiversion. 10.1 LOCKING Whenever a data item is being accessed by a transaction, it must not be modified by any other transaction. In order to ensure this, a transaction needs to acquire a lock

73%	MATCHING BLOCK 231/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)
on the required data items. A lock is a variable associated with each data item			

that indicates whether a read or write operation can be applied to the data item. In addition, it synchronizes the concurrent access of the data item. Acquiring the lock by modifying its value is called locking. It controls the concurrent access and manipulation of the locked data item by other transactions and hence, maintains the consistency and integrity of the database. Database systems mainly use two modes of locking, namely, exclusive locks and shared locks. Exclusive lock (denoted by X) is the commonly used locking strategy that provides a transaction an exclusive control on the data item. A transaction that wants to read as well as write a data item must acquire exclusive lock on the data item. Hence, an exclusive lock is also known as an update lock or a write lock. If a transaction T i has acquired an exclusive lock on a data item, say Q, then T i can both read and write Q. Further, if another transaction, say T j , requests to access Q, then T j has to wait for T i to release its lock on Q. It means that no transaction is allowed to access data item when it is exclusively locked by another transaction. By prohibiting other transactions to modify the locked data item, exclusive lock prevents concurrent transactions from corrupting one another. Shared lock (denoted by S) can be acquired on a data item when a transaction wants to only read a data item and not modify it. Hence, it is also known as read
lock.

71%	MATCHING BLOCK 228/319	SA	Advanced DBMS.pdf (D166063498)

If a transaction T i has acquired a shared lock on data item Q, T i can read but cannot write

on Q. Moreover, any number of transactions can acquire shared locks on the same data item simultaneously 261 without any risk of interference between transactions. However, if a transaction has already acquired a shared lock on the data item, no other transaction can acquire an exclusive lock on that data item. 10.1.1 Lock Compatibility Depending on the type of operations a transaction needs to perform, it should request a lock in an appropriate mode on that data item. If the requested data item is not locked by another transaction, the lock request is granted immediately by the concurrency control manager. However, if the requested data item is already locked, the lock request may or may not be granted depending on the compatibility of the locks. Lock compatibility determines whether locks can be acquired on a data item by multiple transactions at the same time.

67%	MATCHING BLOCK 229/319	SA	Advanced DBMS.pdf (D166063498)

Suppose a transaction T i requests a lock of mode m 1 on a data item Q on which another transaction T j currently holds a lock of mode m 2 . If

mode m 2 is compatible with mode m 1 , the request is immediately granted, otherwise rejected. The lock compatibility can be represented by a matrix called the compatibility matrix (see Figure 10.1). The term “YES” indicates that the request can be granted and “NO” indicates that the request cannot be granted. Requested mode Shared Exclusive Shared YES NO Exclusive NO NO Fig. 10.1 Compatibility matrix If the mode m 1 is shared, the lock request of transaction T j is granted immediately, if and only if m 2 is also shared. Otherwise the lock request is not granted and the transaction T j has to wait. On the other hand, if mode m 1 is exclusive, the lock request (either shared or exclusive) by transaction T j is not granted and T j has to wait. Hence, it is clear that multiple shared-mode locks can be acquired on a data item by different transactions simultaneously. However, an exclusive-mode lock cannot be acquired on a data item until all other locks on that data item have been released. Whenever a lock on a data item is released, the lock status of the data item changes. If the exclusive-mode lock on the data item is released by the transaction, the lock status of that data item becomes unlocked. Now, the lock request of the waiting transaction is considered. If more than one transactions are waiting, the lock request of one of them is granted. If the shared-mode lock is released by the transaction holding the lock on the data item, the lock status of that data item may not always be unlocked. This is due to the reason that more than one shared-mode lock may be held on the data item. The lock status of the data item becomes unlock only when the transaction releasing the lock is the only transaction holding the shared-mode lock on that data item. In order to count the number of transactions holding a shared lock on a data item, the number of transaction is stored in an appropriate data structure along with the data item. The number is increased by one when another transaction is granted a shared lock and is decreased by one when a transaction releases the shared lock. Note that when this number becomes zero, the lock status of the data item becomes unlocked. A data item Q can be locked in the shared mode by executing the statement lock-S(Q). Similarly, Q can be locked in the exclusive mode by executing the statement lock-X(Q). A lock on the data item Q can be released by executing the statement unlock(Q). If a transaction already holds a lock on any data item, we assume that it will not request a lock on that data item again. In addition, if a transaction does not hold a lock on a data item, we assume that it will not unlock a data item. 10.1.2 Implementation of Locking A lock can be considered as a control block, which has the information about the nature of the locked data item and the identity of the transaction that acquires the lock. The locking or unlocking of the data items is implemented by a subsystem of the database system known as lock manager. It receives the lock requests from transactions and replies them with lock grant message or rollback message (in case of deadlock). In response to an unlock request, the lock 262 manager only replies with an acknowledgement. In addition, it may also result in lock grant messages to other waiting transactions. To select a transaction from the list of waiting transactions, the lock manager has to follow some priority technique. Otherwise, it may result in starvation of the transactions waiting for an exclusive lock. To understand the concept, consider a situation in which a transaction, say T i , has a shared lock on a

33%	MATCHING BLOCK 230/319	SA	BOOK SIZE.docx (D50092775)

data item Q and another transaction, say T j , requests an exclusive lock on the data item Q. Clearly, T j has to wait until the transaction T i releases the shared lock on Q.

Further, suppose in the meanwhile, another transaction T k requests a shared lock on Q. Since the mode of lock request is compatible with the lock held by T i , the lock request is granted to T k . Similarly, there may be a sequence of transactions requesting shared lock on Q. In that case, T j never gets exclusive lock on Q, and is said to be starved, that is, waits indefinitely. T j does not starve if the request of T k is queued behind that of T j , even if the mode of lock request is compatible with the lock held by T i . Learn More Both the lock and unlock request must be implemented as atomic operations in order to prevent any type of inconsistency. For this, multiple instances of the lock manager code may run simultaneously to receive the requests; however, the access to lock table requires some synchronization mechanism such as semaphore. To guarantee freedom from starvation problem, the lock manager maintains a linked list of records for each locked data item in the order in which the requests arrive. Each record of the linked list is used to keep the transaction identifier that made the request and the mode in which the lock is requested. It also records whether the request has been granted. Lock manager uses a hash table known as lock

75%	MATCHING BLOCK 232/319	SA	Advanced DBMS.pdf (D166063498)

table, indexed on the data item identifier, to find the linked list for a data item.

Consider a lock table, shown in Figure 10.2, which contains locks for three different data items, namely, Q 1 , Q 2 , and Q 3 . In addition, two transactions, namely, T i and T j , are shown which have been granted locks or are waiting for locks. Observe that T i has been granted locks on Q 1 and Q 3 in shared mode. Similarly, T j has been granted lock on Q 2 and Q 3 in shared mode, and is waiting to acquire a lock on Q 1 in exclusive mode, which has already been locked by T i . Fig. 10.2 Lock table The lock manager processes the request by the transaction to lock and unlock a data item in the way given here. ? Lock Request: When first request to lock a data item arrives, the lock manager creates a new linked list to record the lock request for the data item. In addition, it immediately grants the lock request of the transaction. 263 However, if the linked list for the data item already exists, it includes the request at the end of the linked list. Note that the lock request will be granted only if the lock request is compatible with all the existing locks and no other transaction is waiting for acquiring lock on this data item. Otherwise, the transaction has to wait. ? Unlock Request: When an unlock request for a data item arrives, the lock manager deletes the record corresponding to that transaction from the linked list for that data item. It then checks whether other waiting requests on that data item can be granted. If the request can be granted, it is granted by the lock manager, and the next record, if any, is processed. If a transaction aborts, the lock manager deletes all waiting lock requests by the transaction. In addition, the lock manager releases all locks acquired by the transaction and updates the records in the lock table. 10.2 LOCK-BASED TECHNIQUES A transaction does not release a lock on the data item as long as it uses that data item. It may release the lock immediately after the final accessing of the data item is done. However, releasing the data item immediately is not always desirable. As an illustration, consider transactions T 1 and T 2 , and data items Q and R with the initial value of 500 units and 1000 units, respectively. T 1 wants to deduct 200 units from the data item R and add 200 units to the data item Q. Whereas T 2 wants to add the values of Q and R. The transactions T 1 and T 2 with lock requests are given in Figure 10.3. Fig. 10.3 Transactions T 1 and T 2 with lock requests If these transactions are executed serially, either T 1 followed by T 2 or vice-versa, T 2 will display the sum 1500. Alternatively, T 1 and T 2 may be executed concurrently. A possible schedule of concurrent execution of T 1 and T 2 is shown in Figure 10.4. This schedule shows the statements issued by the transactions in the interleaved manner. Note that the lock must be granted after the transaction requests the lock but before the transaction executes its next statement. The lock can be granted anywhere in between this interval; however, we are not interested in the exact point of time where the lock is granted. For simplicity, we assume that the lock is granted immediately before the execution of next statement by the transaction. 264
Fig. 10.4 Schedule of T 1 and T 2 In Figure 10.4, the concurrent execution of the transactions results in displaying the sum 1300 units instead of 1500. Observe that the database is in an inconsistent state since 200 units are deducted from data item R but not added to Q. In addition, T 2 is allowed to read the values of Q and R before transaction T 1 is complete. This is because the locks on the data items Q and R are released as soon as possible. Now, suppose that the unlock statements in Figure 10.4 are delayed to the end of the transactions. Then, T 1 unlocks Q and R only after the completion of its actions. So, the database remains in consistent state. However, sometimes delaying the lock until the end of transaction may lead to an undesirable situation, called deadlock. Deadlock is a situation that occurs when all the transactions in a set of two or more transactions are in a simultaneous wait state and each of them is

100%	MATCHING BLOCK 233/319	W

waiting for the release of a data item held by

one of the other waiting transaction in the set. None of the transactions can proceed until at least one of the waiting transactions releases lock on the data item. For example, consider the partial schedule of transactions T 3 and T 4 shown in Figure 10.5. Fig. 10.5 Partial schedule Observe that T 3 is waiting for T 4 to unlock Q, and T 4 is waiting for T 3 to unlock R. Thus, a situation is arrived where these two transactions can no longer continue with their normal execution. This situation is called deadlock. Now, one of these 265
transactions must be rolled back by the system so that the data items locked by that transaction are released and become available to the other transaction. From this discussion, it is clear that if locking is not used or if data items are unlocked too early, a database may become inconsistent. On the other hand, if the locks on data items are not released until the end of transaction, deadlock may occur. Out of these two problems, deadlocks are more desirable, since they can be handled by the database system but inconsistent state cannot be handled. Deadlock handling is discussed in detail in Section 10.8. There is a need that all the transactions in a schedule must follow some set of rules called locking technique. These rules indicate when a transaction may lock or unlock any data item. We discuss several locking techniques in subsequent sections. 10.2.1 Two-Phase Locking Two-phase locking requires that each transaction be divided into two phases. During the first phase, the transaction acquires all the locks; during the second phase, it releases all the locks. The phase during which locks are acquired is growing (or expanding) phase. In this phase, the number of locks held by a transaction increases from zero to maximum. On the other hand, a phase during which locks are released is shrinking (or contracting) phase. In this phase, the number of locks held by a transaction decreases from maximum to zero. Whenever, a transaction releases a lock on a data item, it enters into the shrinking phase, and from this point, it is not allowed to acquire any lock further. So, until all the required locks on the data items are acquired, the release of the locks must be delayed. Thus, the two-phase locking leads to lower degree of concurrency among transactions. This is because a transaction cannot release any data item (which is no longer required), if it needs to acquire locks on additional data items later on. Alternatively, it must acquire locks on additional data items before it actually performs certain action on those data items, in order to release other data items. Meanwhile, another transaction that needs to access any of these locked data items is forced to wait.

59%	MATCHING BLOCK 241/319	SA	BOOK SIZE.docx (D50092775)

The point in the schedule at which the transaction successfully acquires its last lock is called the lock point of the transaction. For example, the transactions

T 1 and T 2 (see Figure 10.3) can be rewritten under two-phase locking as shown in Figure 10.6. 266
Fig. 10.6 Transactions T 1 and T 2 in two-phase locking In Figure 10.6, the statements used for releasing the lock are written at the end of the transaction. However, such statements do not always need to appear at the end of the transaction to retain two-phase locking property. For example, the unlock(R) statement of T 1 may appear

71%	MATCHING BLOCK 234/319	SA	Advanced DBMS.pdf (D166063498)

just after the lock-X(Q) statement and still maintains the two-phase locking property. The

two-phase locking produces the serializable schedules. To verify this, consider a schedule S for a set of transactions T i , T j , ..., T k under two-phase locking. Now assume that S is non-serializable, thus, there must exist a cycle T i → T j → T m ...→ T n → T i in the precedence graph for S. Here, T i → T j indicates that T j acquires a lock on a data item released by T i . Similarly, T m acquires lock on a data item released by T j , and finally T i acquires a lock on a data item released by T n . It means T i acquires a lock on a data item after releasing a lock, and

52%	MATCHING BLOCK 235/319	W	

this is a contradiction that T i is following two-phase locking. Thus, our assumption that S is non-serializable is wrong.

Although, serializability is ensured by two-phase locking, cascading rollback and deadlock is possible under two-phase locking. To avoid cascading rollback,

95%	MATCHING BLOCK 236/319	SA	Advanced DBMS.pdf (D166063498)

a modification of two-phase locking called strict two-phase locking can be used. In the

87%	MATCHING BLOCK 237/319	W

strict two-phase locking, a transaction does not release any of its exclusive-mode locks until

it commits or aborts.
If all the transactions {T 1 , T 2 , ..., T n } participating in a schedule S follow strict two-phase locking, the schedule S is said to be strict schedule. Strict schedules ensure that no other transaction can read or write the data item exclusively locked by a transaction T until it commits. It makes strict schedules recoverable. However, strict two-phase locking can also lead to deadlock. Another more restrictive variation of two-phase locking is known as rigorous two-phase locking. In rigorous

95%	MATCHING BLOCK 238/319	W

two-phase locking, a transaction does not release any of its

locks (both exclusive and shared) until it commits or aborts. Lock Conversion When lock conversions are allowed, the transactions can change the mode of locks from one mode to another on a data item on which it already holds a lock. A transaction can request a lock on a data item many times but it can hold only one lock on the data item at any time. Lock conversion helps in achieving more concurrency. 267
To understand the need of lock conversion, suppose that a transaction T i has locked a data item Q 1 in shared mode. Further, consider a transaction T j given in Figure 10.7. Learn More To achieve more concurrency, a new lock mode referred to as update mode can be introduced. This lock mode is compatible with shared mode but not with other lock modes. Initially, each transaction locks the data item in update mode. Later, it can upgrade the lock mode to exclusive if it needs to modify the data item, or downgrade to shared if it does not. Suppose the transaction T j follows two-phase locking, then it needs the data items Q 1 , Q 2 , and Q 3 to be locked in exclusive mode. Clearly, T j has to wait, since T i has locked the data item Q 1 in shared mode. However, one can easily observe that T j needs exclusive lock on Q 1 only when it updates Q 1 , that is, at the end of its execution. Till then, it just needs a shared lock on Q 1 . Thus, if T j initially acquires shared lock on Q 1 , it may start processing concurrently with T i . Later, when T i has done with Q 1 and released its shared lock, then, T j can easily acquire exclusive lock on Q 1 to update it. This changing of lock mode from shared (less restrictive) to exclusive (more restrictive) is known as upgrading. Similarly, changing the lock mode from exclusive to shared is known as downgrading. Therefore, to achieve more concurrency, two-phase locking is extended to allow conversion of locks. Fig. 10.7 Transaction T j NOTE The lock on a data item can be upgraded in only growing phase. On the other hand, the lock on a data item can be downgraded in only shrinking phase. A transaction that already holds a lock of mode m 1 on Q can upgrade it to a more restrictive mode m 2 , if no other transaction holds a lock conflicting with mode m 2 , otherwise it has to wait. On the other hand, a transaction that already holds a lock of mode m 1 on Q can downgrade it to a less restrictive mode m 2 on Q at any time. 10.2.2 Graph-Based Locking We have discussed two-phase locking, which ensures serializability even when the information about the order in which the data items are accessed is unknown. However, if we know in advance the order in which the data items are accessed, it is possible to develop other techniques to ensure conflict serializability. One such technique is graph-based locking. In this technique, a directed acyclic graph called a database graph is formed, which consists of a set S = {A,B,...,L} of all data items as its nodes, and a set of directed edges. A directed edge from A to B (A → B) in the database graph denotes that any transaction T i must access A before B when it needs to access both A and B. For simplicity, we discuss a simple kind of graph-based locking called the tree-locking. In this all database graphs are tree-structured, and any transaction T i can acquire only exclusive locks on data items. A tree-structured database graph for the set S is shown in Figure 10.8. 268

Fig. 10.8 Tree-structured database graph A transaction T i can acquire its first lock on any data item. Suppose T i requests a lock on data item F, it is granted. After that, T i can lock a data item only if it has already locked the parent of that data item. For example, if transaction T i needs to access L, it has to lock J before requesting lock on L. Note that locking the parent of any data item does not automatically lock that data item. Tree locking allows a transaction to unlock a data item at any time. However, once a transaction releases lock on a data item, it cannot relock that data item. Tree-locking not only ensures conflict serializability but also ensures freedom from deadlock. However, recoverability and cascadelessness are not ensured. To ensure recoverability and cascadelessness, the tree-locking technique can be modified. The modified technique does not allow any transaction to release its exclusive locks until it commits or aborts. Clearly, it reduces the concurrency. Alternatively, we have a way that helps in improving concurrency but ensures only recoverability. In this, for each data item, we record the uncommitted transaction T i that issued the last write instruction to that data item. Further, whenever T j reads any such data item, we ensure that T j cannot commit before T i . To ensure this, we record the commit dependency of T j on all the transactions that issued the last write instruction to the data items that T j reads. If any one of them rolls back, T j must also be rolled back. The advantages of the tree-locking are given here. ? It ensures freedom from deadlock. ? Unlike two-phase locking, transactions can unlock data items earlier. ? It ensures shorter waiting times and greater amount of concurrency. The disadvantage of the tree-locking is that in order to access a data item, a transaction has to lock other data items even if it does not need to access them. For example, in order to lock data items A and L in Figure 10.8, a transaction must lock not only A and L, but also data items C, F, and J. Thus, the number of locks and associated locking overhead including possibility of additional waiting time is increased. 10.3 SPECIALIZED LOCKING TECHNIQUES A static view of a database has been considered for locking as discussed so far. In reality, a database is dynamic since the size of database changes over time. To deal with the dynamic nature of database, we need some specialized locking techniques, which are discussed in this section. 10.3.1 Handling the Phantom Problem 269

Due to the dynamic nature of database, the phantom problem may arise. Consider the BOOK relation of Online Book database that stores information about books including their price. Now, suppose that the PUBLISHER relation is modified to store information about average price of books that are published by corresponding publishers in the attribute Avg_price. Consider a transaction T 1 that verifies whether the average price of books in PUBLISHER relation for the publisher P001 is consistent with the information about the individual books recorded in BOOK relation that are published by P001. T 1 first locks all the tuples of books that are published by P001 in BOOK relation and thereafter locks the tuple in PUBLISHER relation referring to P001. Meanwhile, another transaction T 2 inserts a new tuple for a book published by P001 into BOOK relation, and then, before T 1 locks the tuple in PUBLISHER relation referring to P001, T 2 locks this tuple and updates it with the new value. In this case, average information of T 1 will be inconsistent even though both transactions follow two-phase locking, since new book tuple is not taken into account. The new book tuple inserted into BOOK relation by T 2 is called a phantom tuple. This is because T 1 assumes that the relation it has locked includes all information of books published by P001, and this assumption is violated when T 2 inserted the new book tuple into BOOK relation. Here, the problem is that T 1 did not lock what it logically required to lock. Instead of locking existing tuples of publisher P001, it also needed to restrict the insertion of new tuples having P_ID = “P001”. This can be done by using a general technique known as predicate locking. In this technique, all the tuples (whether existing or new tuples that are to be inserted) that satisfy an arbitrary predicate are locked. In our example, since the attribute that refers to the publisher who has published the book is P_ID, the predicate which locks all the books of P001 publisher is P_ID = “P001”. However, the predicate locking is an expensive technique, so most of the systems do not support predicate locking. These systems yet manage to prevent the phantoms problem by using another technique, called index locking. In this technique, any transaction that tries to insert a tuple with predicate P_ID = “P001” must insert a data entry pointing to the new tuple into the index and is blocked until T i releases the lock. These indexes must be locked in order to eliminate the phantom problem. Here, we take only B + -tree index to maintain simplicity. In order to access a relation, one or more indexes are used. In our example, assume that we have an index on BOOK relation for P_ID. Then, the entry in P_ID will be locked for P001. In this way, the creation of phantoms will be prevented since the creation requires the index to be updated and it also requires an exclusive lock to be acquired on that index. 10.3.2 Concurrency Control in Tree-Structured Indexes Like other database objects, indexes also need to be accessed concurrently by the transactions. Further, the changes in index structure between two accesses by a transaction are acceptable as long as the index structure directs the transaction to the correct set of tuples. We can apply two-phase locking to tree-structured indexes like B + -trees; however, it results in low degree of concurrency.

80%	MATCHING BLOCK 239/319	SA	Advanced DBMS.pdf (D166063498)

This is because searching an index always starts at the root, and if any transaction

locks the root node, then all other transactions with conflicting lock requests have to wait. Thus, we require other techniques for managing concurrent access to indexes. In this section, we present two techniques for concurrency control in tree-structured indexes. The first technique is crabbing, which proceeds in the similar manner as a crab walks, that is, releasing lock on a parent node and acquiring lock on a child node and so on alternately. To understand this technique, consider a transaction T i which wants to insert a tuple in a relation on which a B + -tree index exists. Clearly, T i also needs to insert a key-value (or corresponding entry) in the appropriate node. For this, the transaction T i proceeds as follows: ? T i first locks the root node of tree in shared mode. Then, it acquires a shared lock on the child node, which is to be traversed next, and releases the lock on the parent node. This process is repeated till we traverse down to the appropriate leaf node. ? T i then upgrades its shared lock on leaf node to exclusive lock, and inserts the key-value there, assuming that the leaf node has space for the new entry. Otherwise, the node needs to be split. ? In case of split, T i acquires exclusive-mode lock on its parent, performs the splitting operation (as explained in Chapter 07), and releases its lock on the parent node. 270
Note that in case of split, T i needs to acquire exclusive locks on the nodes while moving up the tree. Thus, there is a possibility of deadlock of T i with any other transaction T j that is trying to traverse down the tree. However, such deadlocks can be handled by restarting T j . An alternative technique uses

92%	MATCHING BLOCK 240/319	SA	Advanced DBMS.pdf (D166063498)

a variant of the B + -tree called B-link tree. In B-link tree, every node of the tree (not just leaf nodes) maintains a pointer that points to its right sibling. This technique differs from crabbing as it allows the transactions to release locks on nodes even before acquiring locks on the child nodes. Thus, allows more transactions to operate concurrently. Now, consider a transaction T i that inserts a key-value in a leaf node and a split occurs. The split causes the redistribution of key-values in the sibling nodes as well as in the parent node. Suppose at that point of time, another transaction T j follows the same path for searching a key-value. Since, the key-values of the node are already redistributed with the right siblings by T i , T j may not find the required key-value in the desired leaf node. However, T j can still complete its searching process by following the pointer to the right sibling. This is the reason for the nodes at every level of the tree to have pointers to right siblings. Note that the search and insertion operation cannot result in deadlock. Performing the deletion operation is very simple one. The transaction traverses down the tree to search the required leaf node and removes the key-value from the leaf node. If there are few key-values left in the node after the deletion operation, it needs to be coalesced (or combined). In case of coalescing, the sibling nodes must be locked in exclusive mode. After coalescing on the same level, an exclusive lock is requested on the parent node to remove the deleted node. At this point, the locks on the nodes that are combined are released. Note that if there are few key-values left in the parent node, it can also be coalesced with its siblings. Once the transaction has combined the two parent nodes, its lock is released. In case of coalescing of nodes during deletion operation, a concurrent search operation may find a pointer to a deleted node, if the parent node is not updated. In such situations, search operation has to be restarted. To avoid such inconsistencies, nodes can be left uncoalesced. If this is done, the B + -tree will contain the nodes with few values, which violates its properties. 10.3.3 Multiple-Granularity Locking Before discussing multiple-granularity locking, one must first understand the concept of locking granularity. Locking granularity is the size of the data item that the lock protects. It is important for the performance of a database system. Coarse granularity refers to large data item sizes such as an entire relation or a database, whereas fine granularity refers to a small data item sizes such as either a tuple or an attribute. If the larger data item is locked, it is easier for the lock manager to manage the lock. The overhead associated with locking the large data item is low since there are fewer locks to manage. In this case, the degree of concurrency is low because the transaction is holding a lock on a large data item, even if it is accessing only a small portion of the data item. Observe that the transaction completes its operations successfully, but at the expense of forcing many transactions to wait. On the other hand, the overhead associated with locking the small data item is considerably high since there are many locks to be managed. However, the degree of concurrency in this case is high, since any number of transactions can acquire any number of locks, which will be managed by the lock manager. A transaction requires lock granularity on the basis of the operation being performed. An attempt to modify a particular tuple requires only that tuple to be locked. At the same time, an attempt to modify or delete multiple tuples requires an entire relation to be locked. Since different transactions have dissimilar requests, it is desirable that the database system provides a range of locking granules called multiple-granularity locking. The efficiency of the locking mechanism can be improved by considering the lock granularity to be applied. However, the overheads associated with multiple-granularity locking can outweigh the performance gains of the transactions. Multiple-granularity locking permits each transaction to use levels of locking that are most suitable for its operations. This implies that long transactions use coarse granularity and short transactions use fine granularity. In this way, long transactions can save time by using few locks on a large data item and short transactions do not block the large data item, when its requirement can be satisfied by the small data item. Note that the size of the data items to be locked influences the level of concurrency. While choosing locking granularity, trade-off between concurrency and overheads should be considered. In other words, the choice is between the loss of 271

concurrency due to coarse granularity and increased overheads of maintaining fine granularity. This choice depends upon the type of the applications being executed and how those applications utilize the database. Since the choice depends upon the type of the applications, it is appropriate for the database system to support multiple-granularity locking. As shown in Figure 10.9, the hierarchy of data items of various sizes can be represented in the form of a tree in which small data items are nested within larger data items. The hierarchy in this figure consists of database DB with three files, namely, F 1 , F 2 , and F 3 . Each file consists of several records as its child nodes. Here, notice the difference between the multiple-granularity tree and tree-locking. In the multiple-granularity tree, each non-leaf node represents data associated with its descendents whereas each node in tree-locking is an independent data item. Learn More To predict an appropriate locking granularity for a given transaction, a technique called lock escalation may be used. In this technique, initially, a transaction starts locking items of fine granularity. But when the transaction has exceeded a certain number of locks at that granularity, it starts obtaining locks at the next higher level of granularity. Whenever a transaction acquires shared-mode or exclusive-mode lock on a node in multiple-granularity tree, the descendants of that node are implicitly locked in the same mode. Consider the scenario given here to clear this point: Suppose T i acquires an exclusive lock on file F 1 in Figure 10.9. In that case, it has an exclusive lock on all the records, that is, R 11 ,...,R 1n of that file. Observe that, in this way, T i has to acquire only a single lock on F 1 instead of acquiring locks on individual records of F 1 . Now, assume that T j requests a shared lock on R 11 of F 1 . Now, T j must traverse from the root of the tree to record R 11 to determine whether this request can be granted. If any node in that path is locked in an incompatible mode, the lock request for R 11 is denied and T j must wait. Further, assume that another transaction T k wants to lock the entire database. This can be done by locking the root of the tree. However, this request should not be granted since T i is holding a lock on file F 1 . In order to determine whether the root node can be locked, the tree has to be traversed to examine every node to see whether any of them is currently locked by any other transaction. This would be undesirable and would defeat the purpose of multiple-granularity locking technique. Fig. 10.9 Multiple-granularity tree Instead, we introduce another type of lock mode, called intention lock, in which a transaction intends to explicitly lock a lower level of the tree. While traversing from the root node to the desired node, all the nodes along the path are locked in intention-mode. In simpler terms, no transaction is allowed to acquire a lock on a node before acquiring an intention-
272
mode lock on all its ancestor nodes. For example, to lock a node R 11 in Figure 10.9, a transaction needs to lock all the nodes along the path from root node to node R 11 in an intention-mode before acquiring lock on node R 11 . To provide higher degree of concurrency, the intention-mode is associated with shared-mode and exclusive-mode. When intention-mode is associated with shared-mode, it is called intention-shared (IS) mode, and when it is associated with exclusive-mode, it is called intention-exclusive (IX) mode. To lock a node in shared mode, all its ancestor nodes must be locked in intention-shared (IS) mode. Similarly, to lock a node in exclusive mode, all its ancestor nodes must be locked in intention-exclusive (IS) mode. NOTE

66%	MATCHING BLOCK 242/319	SA	Advanced DBMS.pdf (D166063498)

Intention-shared (IS) lock is incompatible only with exclusive lock whereas intention-exclusive (IX) lock

is incompatible with both shared and exclusive locks. It is clear from the discussion that the lower level of the tree has to be explicitly locked in the mode requested by the transaction. However, it is undesirable if a transaction needs to access only a small portion of the tree. Thus, another type of lock mode, called shared and intention-exclusive (SIX) mode is introduced. The shared and intention-exclusive mode explicitly locks the sub tree rooted by a node in the shared mode, and the lower level of that sub tree is explicitly locked in exclusive-mode. This mode provides the higher degree of concurrency than exclusive mode because it allows other transactions to access the part of the sub tree that is not locked in the exclusive mode. Relative Privilege of Locking Modes Now, consider Figure 10.10, which shows the relative privilege of the locking modes. Among all the locking modes, the highest privilege is given to the exclusive mode. This is because it does not allow other transactions to acquire any lock on the portion of the tree, which is rooted at the node locked in exclusive mode. In addition, the lower level of the sub tree, rooted by that node, is implicitly locked in the exclusive mode. On the other hand, the lowest privilege is given to the intention-shared mode. The shared mode and the intention-exclusive mode are given same privilege. Fig. 10.10 Relative privilege of the locking modes Consider two lock modes, namely, m 1 and m 2 , such that m 1 is given higher privilege than m 2 . Now, if a lock request of mode m 2 is denied on a data item, then a lock request of mode m 1 on the same data item will also be denied definitely. It 273

implies that if there is "NO" in column of m 2 in the compatibility matrix (see Figure 10.11) for a particular lock request, there will be "NO" in the column of m 1 . Fig. 10.11 Compatibility matrix for different access modes Locking Rules Each transaction T i can acquire a lock on node N in any locking mode (see Figure 10.11) by following certain rules, which ensure serializability. These rules are given here. ? T i must adhere to the compatibility matrix given in Figure 10.11. ? T i must acquire any lock on the root of the tree before acquiring any lock on N. ? T i can acquire S or IS lock on N only if it has successfully acquired either IX or IS lock on the parent of N. ? T i can acquire X, SIX, or IX lock on N only if it has successfully acquired either IX or SIX lock on the parent of N. ? T i can acquire additional locks if it has not released any lock. Observe that this is requirement of two-phase locking. ? T i must release the locks in bottom-up order (that is, leaf-to-root). Therefore, T i can release its lock on N only if it has released all its locks on the lower level of the sub tree rooted by N. 10.4 PERFORMANCE OF LOCKING Normally, two factors govern the performance of locking, namely, resource contention and data contention. Resource contention refers to the contention over memory space, computing time and other resources. It determines the rate at which a transaction executes between its lock requests. On the other hand, data contention refers to the contention over data. It determines the number of currently executing transactions. Now, assume that the concurrency control is turned off; in that case the transactions suffer from resource contention. For high loads, the system may thrash, that is, the throughput of the system first increases and then decreases. Initially, the throughput increases since only few transactions request the resources. Later, with the increase in the number of transactions, the throughput decreases. If the system has enough resources (memory space, computing power, etc.) that make the contention over resources negligible, the transactions only suffer from data contention. For high loads, the system may thrash due to aborting (or rollback) and blocking. Both the mechanisms degrade the performance. Clearly, aborting the transaction degrades the performance as the work done so far by the transaction is wasted. However, performance degradation due to blocking is more subtle as it prevents other transactions to access the data item which is held by blocked transaction. An instance of blocking is deadlock in which two more transactions are in a simultaneous wait state, and are waiting for some data item, which is locked by one of the other transactions. Hence, the transactions are blocked till the time one of the deadlocked transactions is aborted. Practically, it is seen that there are less than 1% of transactions, which are involved in a deadlock and there are comparatively lesser aborts. Thus, the system thrashes mainly due to blocking. Consider how system thrashes due to blocking. Initially, the first few transactions are unlikely to block each other; as a result, the throughput increases. Transactions begin to block each other when more and more transactions execute concurrently. Thus, chances of occurrence of blocking increase with the increase in the number of active transaction. However, the throughput does not increase with the increase in the number of transactions. In fact, there comes a point when including additional transaction decreases the throughput. This is because additional transaction is blocked and 274
competes with other transactions to acquire locks on the data item, which decreases the throughput. At this point, the system thrashes, which is shown in Figure 10.12. Fig. 10.12 Thrashing In order to handle this situation, the database administrator should reduce the number of concurrently executing transactions. It is seen that thrashing occurs when 30% of the currently executing transactions are blocked. So, the database administrator should use this fraction of blocked transaction to check whether the system thrashes or not. In addition, the database administrator can increase the throughput in following ways: ? By locking the data item with smallest granularity. ? By reducing the time for which the transaction can hold lock on a data item. ? By reducing hot spots which are frequently accessed data items that remain in the buffer for long period of time. 10.5 TIMESTAMP-BASED TECHNIQUE So far, we have discussed that the locks with the two-phase locking ensures the serializability of schedules. Two-phase locking generates the serializable schedules based on the order in which the transactions acquire the locks on the data items. A transaction requesting a lock on a locked data item may be forced to wait till the data item is unlocked. Serializability of the schedules can also be ensured by another method, which involves ordering the execution of the transactions in advance using timestamps. Timestamp-based concurrency control is a non-lock concurrency control technique, hence, deadlocks cannot occur. 10.5.1 Timestamps Timestamp is a unique identifier assigned to transactions in the order they begin. If transaction T i begins before transaction T j , T i is assigned a lower timestamp. The priority of a transaction will be higher if it is assigned lower timestamp. More precisely, the older transaction has the higher priority since it is assigned a lower timestamp. The timestamp of a transaction T i is denoted by TS(T i). Timestamp can be considered as starting time of the transaction and it is assigned to each transaction T i in the order in which the transactions are entered in the database system. Suppose a new transaction T j enters in the system after T i . In that case, the timestamp of T i is less than that of T j , which can be denoted as TS(T i) &gt; TS(T j). The timestamp of each transaction can be generated in two simple ways: ? The value of system clock can be used as the timestamp. When a transaction enters the system, it must be assigned a timestamp that is equal to the value (that is, current date/time) of the system clock. It must be ensured that no two transactions are assigned the timestamp values at the same tick of the system clock. ? A logical counter can be used which is incremented each time a transaction is assigned a timestamp. When a transaction enters the system, it must be assigned a value that is equal to the value of the counter. The value of 275

the counter may be numbered as 1, 2, 3, ... A counter has finite number of values, so the value of the counter must be reset periodically to zero when no transaction is currently executing for some short period of time. 10.5.2 The Timestamp Ordering In the timestamp ordering, the transactions are ordered on the basis of their timestamps. It means when two concurrently executing transactions, namely, T i and T j are assigned the timestamp values TS(T i) and TS(T j), respectively, such that TS(T i)&gt;TS(T j), the system generates a schedule equivalent to a serial schedule in which the older transaction T i appears before the younger transaction T j . This is termed as timestamp ordering (TO). Things to Remember Although the timestamp ordering is a non-locking technique in the sense that it does not lock a data item from concurrent access for the duration of a transaction. But in actual, at the time of recording timestamps against the data item, it requires a lock on the data item or on its instance for a small duration. When the timestamp ordering is enforced, the order in which the data item is accessed by conflicting operations in the schedule does not violate the serializability order. In order to determine this, the following two timestamps values must be associated

with each data item Q. ? read_TS(Q): The read timestamp of Q; this is the largest timestamp among all the timestamps of transactions that have successfully executed read(Q). ? write_TS(Q): The write timestamp of Q; this is the largest

timestamp among all the timestamps of transactions that have successfully

executed write(Q). NOTE Whenever new read(Q) or write(Q) operations are executed, read_TS(Q) and write_TS(Q) are updated. A number of implementation techniques based on the timestamp ordering have been proposed for concurrency control techniques, which ensure that all the conflicting operations of the transactions are executed in timestamp order. Three of them, namely, basic timestamp ordering, strict timestamp ordering, and Thomas' write rule are discussed here. Basic Timestamp Ordering The basic timestamp ordering ensures that the transaction T i is executed in the timestamp order whenever T i requests read or write operation on Q. This is done by comparing the timestamp of T i with read_TS(Q) and write_TS(Q). If the timestamp order is violated, the system rolls back the transaction T i and restarts it with a new timestamp. In this situation, a transaction T j , which may have used a value of the data item

written by T i , must be rolled back. Similarly, any other transaction T k , which may have used a value

of the

data item written by T j , must also be rolled back, and so on. This situation is known as cascading rollback,

which is one of the problems with basic timestamp ordering. Basic timestamp ordering operates as follows: 1. Transaction T i requests a read operation on Q: 1. If TS(T i)&gt; write_TS(Q),

the read operation is rejected and T i is rolled back.

This is because another transaction with higher timestamp (that is, younger transaction) has already written the value of Q, which T i needs to read. The transaction T i is too late in performing the required read operation

and any other values it has acquired are expected to be inconsistent with the modified value of

Q. Thus, it is better to rollback T i and restart it with a new timestamp. 2. If TS(T i)≥ write_TS(Q), the read operation is executed, and read_TS(Q) is set to the maximum of read_TS(Q) and TS(T i). This is because T i is younger than the transaction that has written the value of Q, which T i needs to read. If the timestamp of T i is more than the current value of read timestamp (that is, read_TS(Q)), the timestamp of T i becomes the new value of read_TS(Q). 2. Transaction T i requests a write operation on Q: 1. If TS(T i) &gt; read_TS(Q),

100%	MATCHING BLOCK 249/319	SA	U234.pdf (D109498494)
the write operation is rejected and T i is rolled back.			

This is because another transaction with higher timestamp (that is, younger transaction) has already read the value of Q, which 276
T i needs to write. So, writing the value of Q by T i violates the timestamp ordering. Thus, it is better to rollback T i and restart it with a new timestamp. 2. If TS(T i) &gt; write_TS(Q),

100%	MATCHING BLOCK 251/319	SA	U234.pdf (D109498494)
the write operation is rejected and T i is rolled back.			

This is because another transaction with higher timestamp (that is, younger transaction) has already written the value of Q, which T i needs to write. So, the value that T i is attempting to write becomes obsolete. Thus, it is better to rollback T i and restart it with a new timestamp. 3. If TS(T i) ≥ read_TS(Q) and TS(T i) ≥ write_TS(Q), the write operation is executed, and the value of TS (T i) becomes the new value of write_TS(Q). This is because T i is younger than both the transactions that have last written the value of Q and read the value of Q. Consider two transactions, namely, T 5 and T 6 such that TS(T 5)&gt; TS(T 6). One possible schedule for T 5 and T 6 is shown in Figure 10.13. For maintaining simplicity, only the read and write operations of the transactions are shown in this schedule. When T 5 issues read(R) operation, it is observed that TS(T 5) &gt; write_TS(R). This is due to the reason that write_TS(R) = TS(T 6), hence, the read operation of T 5 is rejected and T 5 is rolled back. This rollback is required in basic timestamp ordering but it is not always necessary. Fig. 10.13 Schedule for T 5 and T 6 Basic timestamp ordering executes the conflict operations in timestamp order; hence, it ensures conflict serializability. In addition, it is deadlock-free since no transaction ever waits. However, cyclic restart of a transaction may occur in basic timestamp ordering, if it is repeatedly rolled back and restarted. This cyclic restart of the transaction leads to the starvation

46%	MATCHING BLOCK 250/319	SA	Advanced DBMS.pdf (D166063498)
of the transaction. If the restarting of a transaction occurs several times, conflicting transactions need to be temporarily blocked which allows the transaction to finish.			

Strict Timestamp Ordering The strict timestamp ordering is a variation of basic timestamp ordering. In addition to the basic timestamp ordering constraints, it follows another constraint when the transaction T i requests read or write operations on some data item. This constraint ensures a strict schedule, which guarantees recoverability in addition to serializability. Suppose a transaction T i requests a read or write operation on Q and TS(T i) &lt; write_TS(Q), T i is delayed until the transaction, say T j , that wrote the value of Q has committed or aborted. The strict timestamp ordering is implemented by locking Q that has been written by transaction T j until it is either committed or aborted. Furthermore, the strict timestamp ordering ensures freedom from deadlock, since T i waits for T j only if TS(T i) &lt; TS(T j). Thomas' Write Rule Thomas' write rule is the modification to the basic timestamp ordering, in which the rules for write operations are slightly different from those of basic timestamp ordering. The rules for a transaction T i that request a write operation on data item Q are given here. 1. If TS(T i) &gt; read_TS(Q),

100%	MATCHING BLOCK 253/319	SA	U234.pdf (D109498494)
the write operation is rejected and T i is rolled back.			

This is because another transaction with higher timestamp (that is, younger transaction) has already read the value of Q, which T i needs to write. So, W

MATCHING BLOCK 252/319

the value of Q that T i is producing was previously needed, and it had been assumed that the value would never be produced. 277 2.

If TS(T i) &gt; write_TS(Q), the write operation can be ignored. This is because another transaction with higher timestamp (that is, younger transaction) has immediately overwritten the value of Q, which T i wrote. So, no transaction can read the value written by T i and hence, T i is trying to write an obsolete value of Q. 3. If both the conditions given in points 1 and 2 do not occur, the write operation of T i is executed and write_TS(Q) is set to TS(T i). The main difference between these rules and those of basic timestamp ordering is the second rule. It states that if T i requests a write operation on Q and TS(T i) &gt; write_TS(Q), the write operation of T i is ignored. Whereas, according to basic timestamp ordering, if T i requests a write operation on Q and TS(T i) &gt; write_TS(Q), T i has to be rolled back. Fig. 10.14 A non-conflict serializable schedule Unlike other techniques discussed so far, Thomas' write rule does not enforce conflict serializability; however, it makes use of view serializability. It is possible to generate serializable schedules using Thomas' write rule that are not possible under other techniques like two-phase locking, tree-locking, etc. To illustrate this, consider the schedule shown in Figure 10.14. In this figure, the write operation of T 10 succeeds the read operation and precedes the write operation of T 9 . Hence, the schedule is not conflict serializable. Under Thomas' write rule, the write(Q) operation of T 9 is ignored. Therefore,

75% **MATCHING BLOCK 257/319** **SA** BOOK SIZE.docx (D50092775)

the schedule shown in Figure 10.15 is view equivalent to the serial schedule of T 9

followed by T 10 . Fig. 10.15 A schedule under Thomas' write rule 10.6 OPTIMISTIC (OR VALIDATION) TECHNIQUE All the concurrency control techniques, discussed so far (locking and timestamp ordering) result either in transaction delay or transaction rollback, thereby named as pessimistic techniques. These techniques require performing a check before executing any read or write operation. For instance, in locking, a check is done to determine whether the data item being accessed is locked. On the other hand, in timestamp ordering, a check is done on the timestamp of the transaction against the read and write timestamps of the data item to determine whether the transaction can access the data item. These checks can be expensive and represent overhead during transaction execution as they slow down the transactions. In addition, these checks are unnecessary overhead when a majority of transactions are read-only transactions. This is because the rate of conflicts among these transactions may be low. Therefore, these transactions can be executed without applying checks and still maintaining the consistency of the system by using an alternative technique, known as optimistic (or validation) technique. NOTE Optimistic technique is named so because the transactions execute optimistically and thereby assumes that few conflicts will occur among the transactions. In optimistic concurrency control techniques, it is assumed that the transactions do not directly update the data items in the database until they finish their execution. Instead, each transaction maintains local copies of the data items it requires 278
and updates them during execution. All the data items in the database are updated at the end of the transaction execution. In this technique, each transaction T i proceeds through three phases, namely, read phase, validation phase, and write phase,

100% **MATCHING BLOCK 254/319** **SA** Advanced DBMS.pdf (D166063498)

depending on whether it is a read-only or an update transaction. The execution of transaction T i begins with read phase, and the three timestamp values are associated with it during its lifetime. The three phases are explained here. 1. Read phase: At the start of this phase, transaction T i is associated with a timestamp Start(T i). T i reads the values of data items from the database and these values are then stored in the temporary local copies of the data items kept in the workspace of T i . All modifications are performed on these temporary local copies of the data items without updating the actual data items of the database. 2. Validation phase: At the start of this phase, transaction T i is associated with a timestamp Validation(T i). The system performs a validation test when T i decides to commit. This validation test is performed to determine whether the modifications made to the temporary local copies can be copied to the database. In addition, it determines whether there is a possibility of T i to conflict with any other concurrently executing transaction. In case any conflict exists, T i is rolled back, its workspace is cleared and T i is restarted. 3. Write phase: In this phase, the system copies the modifications made by T i in its workspace to the database only if it succeeds in the validation phase. At the end of this phase, T i is associated with a timestamp Finish(T i). Using the value of the timestamp Validation(T i), the serializability order of the transactions can be determined

83%	MATCHING BLOCK 255/319	SA	Advanced DBMS.pdf (D166063498)

by the timestamp ordering technique. Therefore, the value of timestamp Validation(T

i) is chosen as the timestamp of T i instead of Start(T i). This is because we expect that it provides faster response time, if conflict rates among transactions are indeed low. In addition to transaction timestamp, the optimistic technique requires that the read_set and write_set

34%	MATCHING BLOCK 256/319	W	

of the transaction be maintained by the system. The read_set of the transaction is the set of data items it reads and the write_set of the transaction is the set of data items

it writes.

76%	MATCHING BLOCK 259/319	SA	U234.pdf (D109498494)

For every pair of transactions T i and T j such that TS(T

i)&gt;TS(T j), the generated

89%	MATCHING BLOCK 258/319	SA	Advanced DBMS.pdf (D166063498)

schedule must be equivalent to a serial schedule in which T i appears before T

j . In addition, this pair of transactions must hold one of the following validation conditions. 1. Finish(T i)&gt;Start(T j): This condition ensures that the older transaction T i must complete its all three phases before transaction T j begins its start phase. Thus, this condition maintains serializability order. 2. Finish(T i)&gt;Validation(T j): This condition ensures that T i completes its write phase before T j starts its validation phase. It also ensures that the write_set of T i does not overlap with the read_set of T j . In addition, the older transaction T i must complete its write phase before younger transaction T j finishes its read phase and starts its validation phase. This is due to the reason that writes of T j are not overwritten by writes of T i . Hence, it maintains the serializability order. 3. Validation(T i) &gt; Validation(T j): This condition ensures that T i completes its read phase before T j completes its read phase. More precisely, this condition ensures that the write_set of T i does not intersect with the read_set or

83%	MATCHING BLOCK 262/319	SA	DBMS Block 2.pdf (D149011992)

write_set of T j . Since T i completes its read phase before T j completes its read phase,

T i cannot affect the read or write phase of T j , which also maintains serializability. NOTE The validation conditions ensure that modifications made by the younger transaction, say T j , are not visible to the older transaction say T i . To validate T j , the first condition is checked for each committed transaction T i such that TS(T i) &gt; TS(T j). Only if the first condition does not hold, the second condition is checked. Further, if the second condition is false, the third condition is checked. If any one of the mentioned validation conditions holds, there is no conflict and T j is validated successfully. If none of these conditions holds, there is conflict and the validation of T j fails and it is rolled back and restarted. Observe that the first condition permits T j to see the changes made to data items by T i . The second condition permits T j to n read data items when T i is writing data items. However, since T j does not read any data item modified by T i , there is no conflict. 279
The third condition permits both T i and T j to write data items at the same time, but the sets of data items written by the two transactions cannot overlap. Things to Remember To choose an optimistic or pessimistic concurrency control technique for a transaction depends on the type of transaction. The transactions for which recovery would be risky in case of a failure can be better managed with pessimistic techniques. While the transactions for which it is better to take the risk of failure rather than compromising the efficiency can be better managed with optimistic techniques. No transaction can be allowed to commit if any transaction is being validated. This is because the transaction, which is being validated, might overlook conflicts with regard to the newly committed transaction. Before validating other transactions, the write phase of a validated transaction must also be completed. So that the changes, if any, made by the validated transaction must be applied to the database. It can be ensured that at most one transaction is in its validation/write phase at any time by using a synchronization mechanism such as a critical section. This mechanism may lead to lower degree of concurrency. So, it is necessary to keep these phases as short as possible. However, if the values of temporary local copies of data items have to be copied to the actual data items of the database, this can make the write phase long. Thus, an alternative approach that uses a level of indirection to shorten the write phase may be followed. This approach uses a logical pointer to access any data item and we simply change the logical pointer to point to the temporary local copies of data items in the write phase instead of copying the data items. Fig. 10.16 A schedule under optimistic technique For example, consider the schedule given in Figure 10.16. Suppose that the transaction T 11 starts before T 12 , such that TS(T 11) &gt; TS(T 12). Since the pair of transactions satisfies the validation conditions given earlier, the validation phase will be successful. Note that the values of temporary local copies of data items are copied to the actual data items of the database only after the validation phase of T 12 . Furthermore, this schedule maintains serializability order because T 11 reads the old values of Q and R. Since the values of temporary local copies of data items are copied to the actual data items of the database only after the validation phase of T 12 , cascading rollbacks cannot occur. However, starvation of long transactions can be possible due to the conflicts that occur because of short transactions. This conflict results in restarting the long transaction repeatedly. The starvation must be avoided by temporarily blocking the conflicting transactions so that the long transaction can complete its execution. It is clear that optimistic concurrency control technique has certain overheads like locking technique. Following are the overheads in optimistic technique. ? It maintains read_set and write_set for each transaction. 280
? It checks for conflicts among the transactions. ? It copies the values from the workspace of the transaction to the actual database. ? It repeatedly restarts the transactions, which leads to wastage of work they have done so far. 10.7 MULTIVERSION TECHNIQUE So far, we have discussed that serializability must be maintained when one or more of the transactions need to modify the data item. The serializability can be maintained either by delaying an operation or by aborting the requesting transaction. For example, if the appropriate value of the data item has not been written yet, a read operation may be delayed; or the transaction issuing the read operation must be rolled back in case

89%	MATCHING BLOCK 260/319	SA	Advanced DBMS.pdf (D166063498)

the value that it was supposed to read has already been overwritten. However, these

problems can be avoided by another concurrency control technique, which keeps the old version (or value) of each data item in the system when the data item is updated. This concurrency control technique is known as multiversion technique. In this technique, several versions (or values) of a data item are maintained. When a transaction issues a write operation on the data item Q, it creates a new version of Q, the old version of Q is retained. When a transaction issues a read operation on the data item Q, an appropriate version of Q is chosen by concurrency control techniques in such a way that the serializability of the currently executing transactions be maintained. The main drawback of the multiversion technique is that more memory space is required to keep multiple versions of the data items. However, old versions of the data items have to be maintained for recovery purposes. Out of several proposed multiversion techniques, this section discusses two multiversion techniques, namely, multiversion technique based on timestamp ordering and multiversion two-phase locking. 10.7.1 Multiversion Technique Based on Timestamp Ordering For each version of a data item, say Q i , system maintains the value of the version and associates the following two timestamps. ?

read_TS(Q i): The read timestamp of Q i ; this is the largest timestamp among all the timestamps of transactions that have successfully read Q i . ? write_TS(Q i): The write timestamp of Q i ; this is the timestamp of the transaction that

has written the value of Q i . Consider a data item Q with the most recent version Q i , that is write_TS(Q i) is the largest among all the versions. Further, assume a transaction T i with write_TS(Q i) &gt; = TS(T i). Now when T i issues read(Q) request, the system returns the value of Q i and updates the value of read_TS(Q i) with TS(T i) if read_TS(Q i)&gt;TS(T i). On the other hand, when T i issues a write(Q) request, the following situations may occur. ? read_TS(Q i)&lt;TS(T i), which means any younger transaction has already read the value of Q i . In this situation, T i is rolled back. ? TS(Q i)=write_TS(Q i). In this situation, the contents of Q i are overwritten. ? If none of the above situation holds, a new version, say Q j , of Q is created with read_TS(T i)=write_TS(Q i)=TS(T i). The main advantage of the multiversion timestamp ordering technique is that read requests by the transaction are never blocked. Hence, it is important for typical database systems in which read requests are more frequent than write requests. On the other hand, there are two main disadvantages of this technique, which are given here. ? Whenever a transaction reads a data item, read_TS(Q i) is updated. It results in accessing the disk twice, that is, one for data item and another for updating read_TS(Q i). ? Whenever two transactions conflict, one of them is rolled back (rather than wait) in order to resolve the conflict, which could result in cascading and hence, can be expensive. The multiversion based on timestamp ordering technique does not ensure recoverability and cascadelessness. 10.7.2 Multiversion Two-Phase Locking 281
In addition to read and write lock modes, multiversion two-phase locking provides another lock mode, that is, certify. In order to determine whether these lock modes are compatible with each other or not, consider Figure 10.17. Fig. 10.17 Compatibility matrix for multiversion two-phase locking The term "YES" indicates that, if a transaction T i holds the lock on data item Q with the mode specified in column header and another transaction T j requests to acquire the lock on Q with the mode specified in row header, then the lock can be granted. This is because the requested mode is compatible with the mode of lock held. On the other hand, the term "NO" indicates that the requested mode is not compatible with the mode of lock held, so, the transaction that has requested the lock must wait until the lock is released. Unlike locking technique, in multiversion two-phase locking, other transactions are allowed to read a data item while a transaction still holds an exclusive lock on the data item. This is done by maintaining two versions for each data item. One version, known as certified version, must be written by any committed transaction and second version, known as uncertified version, is created when an active transaction acquires an exclusive lock on a data item. The basic idea behind this technique is that transactions can read only the most recently certified version. Suppose a transaction, T j , requests a shared lock on a data item Q, on which another transaction T i holds an exclusive lock. In this situation, T j is allowed to read the certified version of Q while T i is writing

the value of uncertified version of Q. However, once T i is ready to commit, it must acquire a certify lock on

Q. Since certify lock is not compatible with other locks (see Figure 10.17), T i must delay its commit until there is no transaction accessing certified version of the data item in order to obtain the certify lock. Once T i acquires the certify lock on Q, the value of uncertified version becomes the certified version of Q and the uncertified version is deleted. This technique has an advantage over two-phase locking that many read operations can execute concurrently with a single write operation on a data item, which is not possible under two-phase locking. This technique; however, has an overhead that the transaction may have to delay its commit until the certify locks are acquired on all the data items it has updated. This technique avoids cascading rollbacks because transactions read certified version instead of uncertified version. Whereas, deadlock may occur if a read lock is allowed to upgrade to a write lock, and they must be handled using the some deadlock handling technique. 10.8 DEALING WITH DEADLOCK As discussed earlier, deadlock is a situation that occurs when all the transactions in a set of two or more transactions are in a simultaneous wait state and each of them is

waiting for the release of a data item held by

one of the other waiting transaction in the set. None of the transactions can proceed until at least one of the waiting transactions releases lock on the data item. To get rid of this undesirable situation, system has to rollback some of the transactions involved in the deadlock. Various steps the system can take to deal with deadlock are discussed in this section. Deadlock Prevention Deadlock prevention ensures that deadlock never happens. Generally, this technique is used when the chances of occurring deadlock are high. Following are the approaches to prevent the deadlocks. ? Conservative 2PL: In this approach, each transaction locks in advance all the data items that it needs during its life-time. ? Assigning an order to all the data items: In this approach, an ordering is imposed on all the data items in the database. Each transaction acquires locks on the data items in a sequence consistent with that order. 282
? Using timestamps along with locking: In this approach, each transaction is assigned a priority using a unique timestamp, which determines whether a transaction is allowed to wait or is rolled back. A lower timestamp denotes higher priority and vice-versa. The first two approaches can be easily implemented if all the data items that are to be accessed by the transaction are known at the beginning. However, it is not a practical assumption because it is often difficult to predict what data items are needed by a transaction before it begins execution. In addition, both approaches limit concurrency since a transaction locks many data items that remain unused for a long duration. Thus, the third approach is mainly used for deadlock prevention. To understand the third approach, consider a situation in which a data item Q is locked by a transaction T i and another transaction T j issues an incompatible lock request on Q. In this situation, two deadlock prevention techniques using timestamps have been proposed which are discussed here. ? Wait-die: If T i has a lower timestamp (that is, higher priority), T i is allowed to wait. Otherwise, T i is rolled back (dies). ? Wound-wait: If T i has a lower timestamp (that is, higher priority),

100%	MATCHING BLOCK 264/319	SA	Advanced DBMS.pdf (D166063498)
T j is rolled back (T j is wounded by T			

i); otherwise, T i waits. When a transaction is rolled back and restarted, it retains the same timestamp that it was originally assigned. Both the wait-die and wound-wait techniques have some similarities, which are discussed here. ? Starvation is avoided by both the wait-die and wound-wait techniques. In both the techniques, a transaction with the smallest timestamp is not rolled back. In addition, the new transactions are given higher timestamp than the older transactions. Thus, a transaction that is rolled back repeatedly will eventually become the oldest transaction and has the highest priority. Observe that the oldest transaction has the smallest timestamp. Therefore, it will not be rolled back again and will be granted all the locks that it had requested. ? The request to acquire a lock on a data item held by another transaction does not necessarily involve a deadlock. Therefore, unnecessary rollbacks may occur in both wait-die and wound-wait techniques. In spite of these similarities, there are certain important differences between the wait-die and wound-wait techniques, which are given in Table 10.1. Table 10.1 Wait-die and wound-wait technique Wait-Die Technique Wound-Wait Technique In this technique, the waiting that may be required by a transaction with higher priority (that is, older transaction) could be significantly higher. In this technique, an older transaction gets the greater probability of acquiring a lock on the data item. It never waits for a younger transaction to release the lock on its data item. In this technique, a transaction may be rolled back many times before acquiring the lock on the requested data item. For example, when a younger transaction T i requests a data item Q held by an older transaction T j , then it is rolled back. When it is restarted, it again requests a lock on Q. Now, if Q is still locked by T j , T i will be rolled back again. In this way, T i may be rolled back several times till it acquires lock on Q. Rollbacks in wound-wait technique are less as compared to wait-die technique. For example, when an older transaction T i requests a data item Q held by a younger transaction T j then T j is rolled back. When it is restarted, it requests Q, which is now locked by T i . In this situation, T j waits. Deadlock Detection and Recovery 283

If there is a little chance of interference among the transactions and the transactions are short and require few locks, we use deadlock detection and recovery techniques instead of deadlock prevention. To detect the deadlock, the system maintains a wait-for graph, which consists of nodes and directed arcs. The nodes of this graph represent the currently executing transactions, and there exists a directed arc from one node to another, if the transaction is waiting for another transaction to release a lock. If there exists a cycle in the wait-for graph, it indicates the deadlock in the system. To understand the creation of a wait-for graph, consider four transactions T 13 , T 14 , T 15 , and T 16 , whose partial schedule is given in Figure 10.18. Things to Remember Invoking the deadlock detection algorithm at small intervals will add considerable overhead and if it is invoked at large intervals, the deadlock will not be detected for a long time. A number of factors, such as the frequency of deadlocks, the number of transactions affected by deadlock, waiting time of transactions, etc., may affect the choice of interval. Fig. 10.18 A partial schedule for T 13 , T 14 , T 15 , and T 16 In this schedule, the transaction T 13 is waiting for transactions T 14 to release lock on the data item Q 2 . Similarly, T 14 is waiting for T 16 , and T 15 is waiting for T 14 . For this situation, the wait-for graph is represented in Figure 10.19. Observe that there exists no cycle in this graph, thus, there is no deadlock. Now assume that the transaction T 16 requests a lock on the data item Q 4 held by T 15 , a directed arc is added in the wait-for graph from T 16 to T 15 (see Figure 10.20). The creation of this directed arc results in a cycle in the wait-for graph, thus, a deadlock occurs in the system in which the transactions T 14 , T 15 , and T 16 are involved. Fig. 10.19 Wait-for graph The wait-for graph is periodically examined by the system to check the existence of deadlock. Once the deadlock is detected, there is a need to recover from the deadlock. The simplest solution is to abort some of the transactions involved in the deadlock, in order to allow other transactions to proceed. The transactions chosen to be aborted are called victim transactions. Some criteria should be followed to choose victim transaction, like the transaction with fewest locks, the transaction that has done the minimum work, and so on. 284
Fig. 10.20 Wait-for graph with cycle Another approach for deadlock handling is to specify the time-interval for which a transaction is allowed to wait for acquiring a lock on a data item. If the time-out occurs, the transaction is rolled back and restarted. Thus, in case of deadlock one or more transactions will time-out and roll back automatically, thereby, allowing other transactions to proceed. This approach lies somewhere in between deadlock prevention and deadlock detection and recovery. However, this approach has limited use in practical situations, since there is a chance of occurrence of starvation. SUMMARY 1. Concurrency control techniques are required to control the interaction among concurrent transactions. These techniques ensure that the concurrent transactions maintain the integrity of a database by avoiding the interference among them. 2. Whenever a data item is to be accessed by a transaction, it must not be modified by any other transaction. In order to ensure this, a transaction needs to acquire a lock

73%	MATCHING BLOCK 272/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

on the required data items. 3. A lock is a variable associated with each data item

that indicates whether read or write operations can be applied to it. Acquiring the lock by modifying its value is called locking. 4. Database systems mainly use two modes of locking. They are exclusive locks and shared locks. Exclusive lock provides a transaction an exclusive control on the data item. Shared lock can be acquired on a data item when a transaction wants to only read a data item and not modify it. 5. The locking or unlocking of the data items is implemented by a subsystem of the database system known as lock manager. 6. The lock manager maintains a linked list of records for each locked data item

88%	MATCHING BLOCK 266/319	SA	Advanced DBMS.pdf (D166063498)

in the order in which the requests arrive. It uses a hash table

known as lock table, which is indexed on the data item identifier. 7. Two-phase locking requires that each transaction be divided into two phases. During the first phase, the transaction acquires all the locks; during the second phase, it releases all the locks. The phase during which locks are acquired is a growing (or expanding) phase. On the other hand, a phase during which locks are released is a shrinking (or contracting) phase. 8. In

88%	MATCHING BLOCK 267/319	W

strict two-phase, a transaction does not release any of its exclusive-mode locks until that transaction commits or aborts. 9. In rigorous

95%	MATCHING BLOCK 268/319	W	
two-phase locking, a transaction does not release any of its			

locks (both exclusive and shared) until that transaction commits or aborts. 10. A transaction is also permitted to change the lock from one mode to another on the data item on which it already holds a lock. This is known as lock conversion. 11. In tree locking, a transaction can acquire only exclusive locks on data items. Transactions are allowed to unlock a data item at any time; however, a data item released once cannot be relocked. 12. In predicate locking technique, all the tuples (whether existing or new tuples that are to be inserted) that satisfy an arbitrary predicate are locked to avoid phantom problem. 13. There are two techniques for concurrency control in tree-structured indexes like B + -trees. The first technique is crabbing. The second technique uses

95%	MATCHING BLOCK 269/319	SA	Advanced DBMS.pdf (D166063498)
a variant of the B + -tree called B-link tree, in			

which every node of the tree maintains a pointer that refers to its right sibling. 285
14. Locking granularity is the size of the data item that the lock protects. Locking a large data item such as either an entire relation or a database is termed as coarse granularity, whereas, locking a small data item such as either a tuple or an attribute is termed as fine granularity. 15. In intention lock, a transaction intends to explicitly lock a lower level of the tree. When intention-mode is associated with shared-mode, it is called intention-shared (IS) mode, and when it is associated with exclusive- mode, it is called intention-exclusive (IX) mode. 16. Timestamp-based concurrency control is a non-lock concurrency control technique; hence, deadlocks cannot occur. Timestamp is a unique identifier assigned to each transaction in the order it begins. 17. A number of implementation techniques based on the timestamp ordering have been proposed for concurrency control techniques, which ensure that all the conflicting operations of the transactions are executed in timestamp order. Three of them are basic timestamp ordering, strict timestamp ordering, and Thomas' write rule. 18. In optimistic concurrency control techniques, it is assumed that the transactions do not directly update the data items until they finish their execution. 19. In multiversion technique, several versions (or values) of a data item are maintained. When a transaction issues a write operation on the data item Q, it creates a new version of Q, the old version of Q is retained. 20. In multiversion two-phase locking, other transactions are allowed to read a data item while a transaction still holds an exclusive lock on the data item. 21. Deadlock is a situation that occurs when all the transactions in a set of two or more transactions are in a simultaneous wait state and each of them is

		W	
	MATCHING BLOCK 270/319		
waiting for the release of a data item held by			

one of the other waiting transaction in the set. 22. Different approaches to prevent deadlocks are conservative 2PL, assigning an order to data items and using timestamps along with locking. 23. To detect a deadlock, the system maintains a directed graph called wait-for graph. If there exists a cycle in the wait-for graph, it indicates the deadlock in the system. 24. One of the deadlocked transactions chosen to be aborted is termed as the victim transaction. 286 CHAPTER 11 DATABASE RECOVERY SYSTEM After reading this chapter, the reader will understand: ? Types of failures that can occur in a system ? Caching of disk pages containing data items ? How system log is maintained during normal execution so that the recovery process can be performed ? The process of check pointing, which

100%	MATCHING BLOCK 271/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)
helps in reducing the time taken to recover from a crash ?			

The two categories of recovery techniques, namely, log-based recovery techniques and shadow paging ? The further classification of log-based techniques into two types, namely, techniques based on deferred update and techniques based on immediate update ? Recovery of a system with multiple concurrent transactions ? ARIES recovery algorithm ? Recovery of a system from catastrophic failures In every computer system, there is always a possibility of occurrence of a failure, which may result in loss of information. Therefore, if a failure occurs, then it is the responsibility of database management system (DBMS) to recover the database as quickly, and with as little damaging impact on users, as possible. The main aim of recovery is to restore the database to the most recent consistent state. The recovery manager component of DBMS is responsible for performing the recovery operations. The recovery manager ensures that the two important properties of transactions, namely, atomicity and durability are preserved. It preserves atomicity by undoing the actions of uncommitted transactions and durability by ensuring that all the actions of committed transactions survive any type of failure. The recovery manager keeps track of all the changes that were applied to the data by various transactions and thus, deals with various states of the database. The recovery manager must also provide the high availability to its users, which requires the minimum levels of services and robustness in case of failures. The recovery depends on the type of failure and the files of the database affected by the failure. This chapter first discusses the various types of failures that can cause abnormal termination of transactions, and then discusses the steps that the system takes during normal execution of transactions for recovery purpose. It then discusses how recovery is performed in case of system crashes. Finally, it discusses how the data is recovered in case of disk failures and natural disasters. 11.1 TYPES OF FAILURES There are several types of failures that can occur in a system and stop the normal execution of the transactions. Each of them is handled in a different manner. Let us discuss these failures. ? Logical error: Transactions may fail due to a logical error in the transaction such as incorrect input, integer overflow, division by zero, etc. Certain exceptional conditions that are not programmed correctly may also result in cancellation of the transaction. For example, insufficient balance in a customer account results in the failure of fund withdrawal transaction. If these types of exceptions are checked within the transaction itself, they do not result in transaction failure. ? System error: Any undesirable state of the system such as deadlock, incorrect synchronization, etc., may also stop the normal execution of the transactions. ? Computer failure (system crash): Any type of hardware malfunctioning such as RAM failure, error in application software, bug in operating system, and network problem can also bring the transaction processing to a halt. It is assumed that such a failure results in the loss of data in volatile storage and does not affect the contents of the non-volatile storage media such as disks. The assumption that the hardware and software errors bring only the system to a halt and has no effect on the contents of the non-volatile storage media is known as fail-stop assumption. 287
? Disk failure: Disk failure, also known as media failure, refers to the loss of data in some disk blocks because of disk read/write head crash, power disruption or error in data transfer operation. These errors also cause the abnormal termination of the transaction if occurred during a read/write operation of the transaction. ? Physical problems and environment disasters: The physical problems include theft, fire, sabotage, accidental overwriting of secondary storage media, etc. The environment disasters include floods, earthquakes, tsunami, etc. NOTE The transaction failure and system crash are more common types of failures as compared to disk failure and natural disasters. In case of a system crash, the information residing in the volatile storage such as main memory and cache memory is lost. Therefore, transaction failure and system crash are non-catastrophic failures as they do not result in the loss of non- volatile storage. On the other hand, disk failure and environment disasters are catastrophic failures as they result in the loss of non-volatile storage. 11.2 CACHING OF DISK PAGES Whenever a transaction needs to update the database, the disk pages (or disk blocks) containing the data items to be modified are first cached (buffered) by the cache manager into the main memory and then modified in the memory before being written back to the disk. A cache directory is maintained to keep track of all the data items present in the buffers. When an operation needs to be performed on a data item, the cache directory is first searched

75%	MATCHING BLOCK 273/319	SA	47F417BA15858476660.pdf (D123781188)

to determine whether the disk page containing the data item resides in the cache. If it is not

present in the cache, the data item is searched on the disk and the appropriate disk page is copied in

70%	MATCHING BLOCK 274/319	SA	47F417BA15858476660.pdf (D123781188)

the cache. Sometimes it may be necessary to replace some of the disk pages to create space for the new pages. Any page-replacement strategy such as least recently used (LRU) or first-in-first- out (FIFO) can be used for replacing the disk page. Each memory buffer has a free bit associated with it which indicates whether the buffer is free (available for allocation) or not. Other associated bits are dirty bit and pin/unpin bit. The data which is modified in the cache can be copied back to the disk using either of the two strategies, namely, in-place updating and shadow paging. In in-place updating strategy, single copy of the data items are maintained on the disk. The updated buffer is written back to the same original disk location and thus, the old value of any updated data item on the disk is overwritten by the new value. However, in shadow paging, multiple copies of the data items can be maintained on the disk. The updated buffer is written at a different location on the disk and thus, the old value of the data item is not overwritten by the new value.

100%	MATCHING BLOCK 275/319	SA	47F417BA15858476660.pdf (D123781188)

Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a

modified page in the cache buffer can be written back to the database on disk. ? Steal: In this approach, an updated cache page may be written to the disk before the transaction commits. It gives more freedom in selecting buffer pages to be replaced by the cache manager. ? No-steal: In this approach, an updated cache page cannot be written to the disk before the transaction commits. The pin bit is set until the updating transaction commits. ? Force: In this approach, all the updated cache pages are immediately written (force-written) to disk when the transaction commits. ? No-force: In this approach, the updated cache pages are not necessarily written to disk immediately when the transaction commits. Things to Remember Steal/no-force approach is the most desirable approach in practice as it gives maximum degree of freedom to the cache manager in selecting the victim pages for replacement, and in scheduling the writes to disk. 11.3

99%	MATCHING BLOCK 277/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

RECOVERY RELATED STEPS DURING NORMAL EXECUTION When a DBMS is restarted after a crash, the control is given to the recovery manager, which is responsible to bring the database to a consistent state. To enable it to perform its task in the event of failure, the recovery manager maintains some information during the normal execution of transactions. It maintains a system log of all the modifications to the 288 database and stores it on the stable storage, which is guaranteed to survive failures. The stable storage is implemented by storing the database on a number of separate non-volatile storage devices such as disks or tapes (perhaps in different locations). Information residing in stable storage is assumed to be never lost (in real life never cannot be guaranteed). The amount of work involved during recovery depends on the changes made by the committed transactions that have not been written to the disk at the time of crash. To reduce the time to recover from a crash, the DBMS periodically force- write all the modified buffer pages during normal execution. A process called checkpointing also helps in reducing the time taken to recover from a crash. 11.3.1

The System Log During the normal transaction processing, the system stores relevant information in a log to make sure that enough information is available to recover from failures. A log is a sequence of log records that contains essential data for each transaction which has updated the database. The various types of log records that are maintained in a log are listed here. ? Start record: The log record [T i , start] is used to indicate that the transaction T i has started. ? Update log record: The log record [T i ,X,V o ,V n] is used to indicate that the transaction T i has performed an update operation on the data item X, having the old value V o and new value V n . ? Read record: The log record [T i ,X] is used to indicate that the transaction T i has read the data item X from the database. ? Commit record: The log record [T i ,commit] is used to indicate that the transaction T i ? Abort record: The log record [T i ,abort] is used to indicate that the transaction T i has aborted, and needs to be rolled back. Whenever a transaction needs to update a data item, two types of log entry information, namely, UNDO-type log entry and REDO-type log entry are maintained.

42%	MATCHING BLOCK 278/319	SA	47F417BA15858476660.pdf (D123781188)

The UNDO-type log entry includes the before-image (BFIM) of the data item which is used to undo the effect of an operation on the

database. A REDO-type log entry, on the other hand, includes the after-image (AFIM) of the data item which is used to redo the effect of an operation on the database, in case the system fails before reflecting the new value of the data item to the database. A BFIM is a copy of the data item prior to modification and an AFIM is a copy of the data item after modification. Before making any changes to the database, it is necessary to force-write all log records to the stable storage. This is known as write-ahead logging (WAL). In general, write-ahead logging protocol states that 1. A transaction is not allowed to update a data item
in the database on the disk until all its UNDO-type log records

62%	MATCHING BLOCK 276/319	W

have been force-written to the disk. 2. The transaction is not allowed to commit until all its REDO-type and UNDO-type log records have been force- written to

the disk. If the system fails, we can recover the consistent state of the database by examining the log and using one of the recovery techniques that are discussed in Section 11.4. WAL is required only for in-place updating. In shadowing, since both BFIM and AFIM of the data item to be modified are kept on the disk, it is not necessary to maintain a log for recovery. To understand how log records are maintained for a transaction, consider

65%	MATCHING BLOCK 279/319	SA	DBMS_AIML_FINAL.pdf (D111167082)

a transaction T 1 that transfers $100 from account A to account B. Suppose accounts A and B

have initial values $2000 and $1500, respectively. The transaction T 1 is given here. T 1 :

100%	MATCHING BLOCK 280/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

read(A); A:= A - 100; 289 write(A); read(B); B:= B + 100; write(B); The

log records for the transaction T 1 are shown in Figure 11.1. For now, we assume that each time a log record for a transaction is created it is immediately written to the stable storage. However, writing each log record individually imposes an extra overhead on the system. To reduce this overhead, multiple log records are first collected in the log buffer in main memory and then copied to the stable storage in a single write operation. This is called log record buffering. Thus, a log record resides in the main memory for some time until it is written to the stable storage. The process of writing the buffered log to disk is known as log force. [T 1 , start] [T 1 ,A] [T 1 , A, 2000, 1900] [T 1 ,B] [T 1 , B, 1500, 1600] [T 1 , commit] Fig. 11.1 Log records for transaction T 1 The order of log records in the stable storage must be exactly the same as that in the log buffer. Note that during transaction rollback, only write operations are undone. The

95%	MATCHING BLOCK 281/319	SA	47F417BA15858476660.pdf (D123781188)

read operations are recorded in the log only to determine whether

the cascading rollback is required or not. In practice, it is very complex and time-consuming to handle cascading rollback. Therefore, the recovery algorithms are designed in such a way that cascading rollback is never required.

92%	MATCHING BLOCK 283/319	SA	47F417BA15858476660.pdf (D123781188)

Hence, there is no need to record any read operations in the log.

From now onwards, we will show the log records without the read operations. 11.3.2 Checkpointing When the system restarts after a failure, the entire log needs to be scanned to determine the transactions that need to be redone and undone. The main disadvantage of this approach is that scanning of entire log is time-consuming. Moreover, the committed transactions that have already written their updates on the database also need to be redone as there is no way to find out the extent to which the changes of committed transactions have been propagated to the database. This causes recovery to take longer time. To reduce the time for recovery by avoiding redo of the unnecessary work, checkpoints are used.

88%	MATCHING BLOCK 282/319	W
The checkpoint approach is an additional component of the logging scheme. A checkpoint		

is periodically written into the log. It is typically written at a point when the system writes out all the modified buffers to the database on the disk. As a result, the transactions that have committed prior to the checkpoint will have their [T i , commit] record before the [checkpoint] record. The write operations of these transactions need not be

58%	MATCHING BLOCK 286/319	SA	47F417BA15858476660.pdf (D123781188)
redone in case of a failure as all updates are already recorded in the database on disk.			

When a checkpoint is taken, the following actions are performed. 1. The execution of the currently running transactions is suspended. 2. All the log records currently residing in the main memory are written to the stable storage. 3. All the modified buffer blocks are force-written to the disk. 4. A [checkpoint] record is written to the log on the stable storage. 5. The execution of suspended transactions is resumed. If the number of blocks in the buffer are large, force-writing of all these pages may take long time to finish. The time taken to force-write all modified buffers can result in undesirable delay in the processing of current transactions. For reducing this delay, a technique called fuzzy checkpointing is used in practice. In fuzzy checkpointing, a [checkpoint] record (known as fuzzy checkpoint) is written in the log before writing the modified buffer blocks to the disk. The system is then allowed 290
to resume the execution of other transactions without having to wait for the completion of step 3. However, a system failure may occur while modified buffer blocks are being written to the disk. In that case, the fuzzy checkpoint would no longer be valid. Thus, the system is required to keep track of the previous checkpoint for which all the actions (1 to 5) have been successfully performed. For this, the system maintains a pointer to that checkpoint and it is updated to point to new checkpoint only when all the modified buffer blocks have been written to the disk. The transactions committed before the checkpoint time need not be considered during recovery process. This reduces the amount of work required to be done during recovery. 11.4 RECOVERY TECHNIQUES The recovery techniques are categorized mainly into two types, namely, log-based recovery techniques and shadow paging. The log-based recovery techniques maintain transaction logs to keep track of all update operations of the transactions. On the other hand, the shadow paging technique does not require the use of a log as both the after image and before image of the data item to be modified are maintained on the disk. This section discovers the recovery techniques for the situations where transactions execute serially one after the other and only one transaction is active at a time. Recovery in situations where multiple transactions execute concurrently is discussed in Section 11.5. 11.4.1 Log-based Recovery The log-based recovery techniques are classified into two types, namely, techniques based on deferred update and techniques based on immediate update. In deferred update technique, the transaction is not allowed to update the database on disk until the transaction enters into the partially committed state. In immediate update technique, as soon as a data item is modified in cache, the disk copy is immediately updated. That is, the transaction is allowed to update the database in its active state. To recover from a failure, basically two operations, namely, undo and redo are applied with the help of the log on the last consistent state of the database. The undo operation reverses (rolls back) the changes made to the database by an uncommitted transaction and restores

100%	MATCHING BLOCK 284/319	SA	Advanced DBMS.pdf (D166063498)
the database to the consistent state that existed before the			

start of transaction. Learn More Undoing the effect of only some (not all) uncommitted transctions is known as partial undo. Similarly, redoing the effect of only some committed transactions is known as partial redo. The redo operation reapplies the changes of a committed transaction and restores the database to the consistent state it would be at the end of the transaction. The redo operation is required when the changes of a committed transaction are not or partially reflected to the database on disk. The redo modifies the database on disk to the new values for the committed transaction. Sometimes the undo and redo operations may also fail due to any reason,

100%	MATCHING BLOCK 285/319	W
and this type of failure can occur any number of times before the recovery is		

completely successful. Therefore, the undo and redo operations for a given transaction are required to be idempotent, which implies that executing any of these operations several times must be equivalent to executing it once. That is, undo(operation)=undo(undo(...undo(operation)...)) redo(operation)=redo(redo(...redo(operation)...)) Recovery Techniques Based on Deferred Update The deferred update technique records all update operations of the transactions in the log, but postpones the execution of all the update operations until the transactions enter into the partially committed state. Once the log records are written to the stable storage under WAL protocol, the database on the disk is updated. 291
If the

67%	MATCHING BLOCK 287/319	SA	47F417BA15858476660.pdf (D123781188)

transaction fails before reaching its commit point no undo is required as the transaction has not affected the database on

the disk. In this case, the information in the log is simply ignored and the failed

96%	MATCHING BLOCK 289/319	SA	47F417BA15858476660.pdf (D123781188)

transaction must be restarted either automatically by the recovery process or manually by the user.

However, redo operation

89%	MATCHING BLOCK 288/319	W	

is required in case the system fails after the transaction commits but before all

the updates are reflected in the database on disk. In such situation, the log is used to redo all the transactions affected by this failure. Thus, this is known as NO-UNDO/REDO recovery algorithm. Note that the update log records, in this case, only contain the AFIM of the data item to be modified. The BFIM of the data item is omitted, as no undo operation is required in case of a failure. This technique uses no-steal/no-force approach for writing the modified buffers to the database on disk. For example, consider the transactions T 1 and T 2 given in Figure 11.2(a). The transaction T 1 transfers $100 from account A to account B and transaction T 2 deposits $200 to account C. Assume that the initial values of account A, B, and C are $2000, $1500, and $500, respectively. Suppose that these two transactions

100%	MATCHING BLOCK 290/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

are executed serially in the order T 1 followed by T 2 .

The corresponding log records (only for write operations) are shown in Figure 11.2(b). If the system fails before writing the log record [T 1 , commit], no redo is required as no commit record is found in the log. The values of account A and B remain $2000 and $1500, respectively. If the system fails after writing the log record [T 1 , commit], but before writing the commit record for the transaction T 2 , only redo(T 1) operation is performed. If the system fails after writing the log record [T 2 , commit], both T 1 and T 2 need to be redone. In general, for each commit record [T i , commit] found in the log, the operation redo(T i) is performed by the system. Fig. 11.2 Transactions and their log records Recovery Techniques Based on Immediate Update The immediate update technique allows a transaction to update the database on the disk immediately, even before it enters into partially committed state. That is, a transaction can update the database while it is in the active state. The data modifications made by the active transactions are known as uncommitted modifications. Like deferred update technique, immediate update technique also records the update operations in the log (on the stable storage) first so that the database can be recovered after a failure. If the system fails before the transaction enters into partially committed state, all the changes made to the database by this transaction must be undone. However, if immediate update ensures that all the updates made by
a transaction

100%	MATCHING BLOCK 296/319	SA	Sem III_ BCA_B21CA05DC.docx (D165628090)

are recorded in the database on the disk before the transaction

commits,
redo operation is not required. Therefore, this 292
recovery algorithm is known as UNDO/NO-REDO recovery algorithm. This technique uses steal/force approach for writing the modified buffers to the database on the disk. The log records in this case contain only before-image of the data item. The after-image can be omitted, since no redo is required. The undo operation is performed by replacing the current value of a data item in the database by its BFIM stored in the log. For example, consider the transaction T 1 and T 2 given in Figure 11.2(a). The corresponding log records (only for write operations) for these transactions are shown in Figure 11.3. [

T 1 , start] [T 1 , A, 2000] [T 1 , B, 1500] [T 1 , commit] [T 2 , start] [T 2 , C, 500] [T 2 , commit]

Fig. 11.3 Log records for transaction T 1 and T 2 without AFIMs If the system fails before writing the log record [T 1 , commit], the operation undo(T 1) is performed. If the system fails after writing the log record [T 1 , commit] but before writing the commit record for the transaction T 2 , the operation undo(T 2) is performed. The operation redo(T 1) is not required as it is assumed that all updates of transaction T 1 are already reflected to the database on the disk. If the system fails after writing the log record [T 2 , commit], neither undo nor redo operation is required. There is a possibility that immediate update allows a transaction

to commit before all its changes are written to the database. In

this case, both undo and redo operations may be required. Therefore, this algorithm is known as UNDO/REDO recovery algorithm. This technique uses steal/no-force approach for writing the modified buffers to the database on the disk. The UNDO/REDO algorithm is the most commonly used algorithm. The log records in this case contain both the BFIM and AFIM of the data item to be modified. For example, consider the transaction T 1 and T 2 given in Figure 11.2(a). The corresponding log records (only for write operations) containing both BFIM and AFIM of the data items to be modified are shown in Figure 11.4. If the system fails before writing the log record [T 1 , commit], the operation undo(T 1) is performed. If the system fails after writing the log record [T 1 , commit] but before writing the commit record for the transaction T 2 , the operations redo(T 1) and undo(T 2) are performed. Here, the transaction T 1 should be redone as all updates of T 1 may not be reflected to the database on the disk. If the system fails after writing the log record [T 2 , commit], the operations redo(T 1) and redo(T 2) need to be performed. [

T 1 , start] [T 1 , A, 2000, 1900] [T 1 , B, 1500, 1600] [T 1 , commit] [T 2 , start] [T 2 ,C,500,700] [T 2 , commit]

Fig. 11.4 Log records for transactions T 1 and T 2 with both BFIMs and AFIMs 11.4.2 Shadow Paging The shadow paging technique is different from the log-based recovery techniques in a way that it does not require the use of log in an environment where only one transaction is active at a time. However, in an environment where multiple concurrent transactions are executing, the log is maintained for the concurrency control method. We will not discuss shadow paging in an environment with multiple concurrent transactions. 293
Shadow paging is based on making copies (known as shadow copies) of

the database. This technique assumes that only one transaction is active at a time

and thus, can be used as an alternative to log-based recovery technique in case the transactions are executed serially. This recovery technique is simple to implement but not as efficient as log-based recovery techniques.

Shadow paging considers the database to be made up of fixed-size logical units of storage called pages. These pages are mapped into physical blocks of storage with the help of page table (or directory). The physical blocks are of the same size as that of the logical blocks. A page table

with n entries is constructed in which the i th entry in the page table points to the i th database page on the

disk. The main idea behind this technique is to maintain two page tables, namely, current page table and shadow page table during the life of a transaction. The entries in the current page table (or current directory)
points to the most recent database pages on the disk.

When a transaction starts, the current page table is copied into a shadow page table (

or shadow directory). The

shadow page table is then saved on the disk and the current page table is

used by the transaction. The shadow page table is never modified during the execution of the transaction. Initially, both the tables are identical. Whenever a write operation is to be performed on any database page, a copy of this page is created onto an unused page on the disk. The current page table is then made to point to this copy, and the update is performed on this copy. The shadow page table continues to point to the old unchanged page. The old copy of the page is never overwritten. This implies that for the

pages updated by the transactions, two versions are kept. The

shadow page table points to the old version and the current page table points to the new version of the page. Figure 11.5 illustrates the concept of shadow and current page table. In this figure, pages 1 and 3 have two versions. The shadow page table references the old versions and current page table references the new versions of page 1 and 3. Fig. 11.5 Shadow paging In case of a failure, the current page table is discarded and the disk blocks containing the modified disk pages are released. The previous state of the database prior to the transaction execution can be recovered from the shadow page table. In case no failure occurs and the transaction is committed, the shadow page table is discarded and the disk blocks with the old data are released. Since recovery in this case requires neither undo nor redo operations, this algorithm is known as NO-UNDO/NO-REDO recovery algorithm. This technique uses no-steal/force approach for writing the modified buffers to the database on the disk. The shadow paging technique has several disadvantages. Some of them are discussed here. ? Data fragmentation: Shadow paging technique causes the updated database pages to change locations on the disk. This makes it difficult to keep the related database pages together on the disk. ? Garbage collection: Whenever a transaction commits, the database pages containing the old version of data are considered as garbage as they do not contain any usable information. Thus, it is necessary to find all of the 294
garbage pages periodically and add them to the list of free pages. This process of finding garbage pages periodically is known as garbage collection. Though this process incurs an extra overhead on the system, but is necessary to improve the system performance. ? Harder to extend: It is difficult to extend the algorithm to allow transactions to run concurrently. 11.5 RECOVERY FOR CONCURRENT TRANSACTIONS The recovery techniques discussed so far handle the transactions in an environment where transactions execute serially. This section discusses how the log-based recovery algorithms can be extended to handle multiple transactions running concurrently. Note that regardless of the number of transactions, the system maintains a single log for all the transactions. In case, where several transactions execute concurrently, the recovery algorithm may be more complex, depending on the concurrency control technique being used.

In general, the greater the degree of concurrency, the more time-consuming the task of recovery becomes. Consider a system in which strict two-phase locking protocol is used for concurrency control. To minimize the work of a recovery manager, checkpoints are also maintained at regular intervals. Two lists, namely, active list and commit list are maintained by the recovery system. All the active transactions T A are entered in the active list and all the committed transactions T C since the last checkpoint are entered in the commit list. First, consider the case when UNDO/REDO recovery algorithm is used. During recovery process,

the write operations of all the transactions in the commit list are redone in the order in which they were written to the log. The

write operations of all the transactions in the active list are

undone in the reverse of the order in which they were written

to the log.
For example, consider five transactions T 1 , T 2 , T 3 , T 4 , and T 5 executing concurrently as shown in Figure 11.6. Suppose a checkpoint is made at time t c and the system crash occurs at time t f . During the recovery process, transactions T 2 and T 3 (present in commit list) need to be redone, as they are committed after the last checkpoint. Since the transaction T 1 is committed before the last checkpoint, its operations need not be redone. The transactions T 4 and T 5 (present in the active list) were not committed at the time of system crash and hence, need to be undone. If NO-UNDO/REDO algorithm is followed, the transactions in the active list are simply ignored, as these transactions have not affected the database on the disk and hence, need not be undone. Similarly, if UNDO/NO-REDO technique is followed, the transactions in the commit list are ignored as their effects are already reflected in the database on disk. Fig. 11.6 Recovery with concurrent transactions In case a data item Q is modified by different transactions, all the updates made to Q are overwritten by the last write operation. The NO-UNDO/REDO algorithm can be made more efficient by only redoing the last update of Q, rather than redoing all the update operations. In this situation, the log is scanned in the backward direction, starting from the end. As soon as a write operation on a data item Q is redone, Q is added in a list of redone items. Now, before applying a redo 295
operation on any data item, it is first searched in the list of redone items.

If the item is found in the list, it is not redone again as its last value has already been recovered. 11.6

ARIES RECOVERY ALGORITHM Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is an example of recovery algorithm which is widely used in the database systems. It uses steal/no-force approach for writing the modified buffers back to the database on the disk. This implies that ARIES follows UNDO/REDO technique. The ARIES recovery algorithm is based on three main principles which are given here. ? Write-ahead logging: This principle states that before making any changes to the database, it is necessary to force-write the log records to the stable storage. ? Repeating history during redo: When the system restarts after a crash, ARIES retraces all the

actions of database system prior to the crash to bring the database to the state which existed at the time of the crash.

It then undoes the actions of all the

transactions that were not committed at the time of the crash. ? Logging changes during undo:

A separate log is maintained while undoing a transaction to make sure that the undo operation once completed is not repeated in case the failure occurs during the recovery itself, which causes restart of the recovery process. In ARIES, each log record is assigned a unique id called the log sequence number (LSN) in monotonically increasing order. This id indicates the address of the log record on the disk. The ARIES algorithm splits a log into a number of sequential log files, each of which is assigned a file number. When a particular log file grows beyond a certain limit, a new log file is created and new records are added to that

file. The new file is assigned a number higher by 1 than the previous log file.

In this case,

the LSN consists of a file number and an offset within the file.

Every data page has a pageLSN field that is set to the LSN of the log record corresponding to the last update on the page. During the redo operation, the

log records with LSN less than or equal to pageLSN of the page should not be executed as their actions are already reflected

in the database. The set of all log records for a particular transaction are stored in the form of linked list. Every log record includes some common fields such as previous LSN (prevLSN), transaction ID, and type of the log record. The prevLSN contains the address of the previous log record of the transaction. It is required to traverse the linked list in backward direction. The transaction ID field contains the id of the transaction for which the log record is maintained. The type field contains the type of the log record. The update log record also contains some additional fields such as pageID containing the data item, length of the updated item, its offset from the beginning of the page, and the BFIM and AFIM of the data item being modified. In addition to the log, two tables, namely, transaction table and dirty page table are also maintained for efficient recovery. The transaction table contains a record for each active transaction. The record contains the information such as transaction ID, transaction status (active, committed, or aborted), and the LSN of the most recent log record (called lastLSN) for the transaction. The dirty page table contains a record for each dirty page in the buffer. Each record consists of a pageID and recLSN. The recLSN is the LSN of the first log record that has updated the page. This LSN gives the earliest log record that might have to be redone for this page when the system restarts after a crash. During normal operation, the transaction table and the dirty page table are maintained by the transaction manager and buffer manager, respectively. The most recent portion of the log, which is kept in main memory, is called the log tail. The log tail is periodically forced- written to the stable storage. A log record is written for each of these actions. ? updating a page (write operation) ? committing a transaction ? aborting a transaction ? undoing an update ? ending a transaction The log records for update operation, and committing and aborting a transaction have already been discussed. For undoing an update operation, a compensation log record (CLR) is maintained that records the actions taken during the 296
rollback of a particular update operation. It is appended to the log tail as any other log record. The CLR contains an additional field, called the UndoNextLSN, which contains the LSN of the log record

that needs to be undone next, when the transaction is being rolled back.

Unlike update log records, the CLRs describe the actions that have already been undone and hence, will not be undone again because we never need to undo an action that has already been undone. The number of CLRs written during undo is same as of the number of update log records for transactions that were not committed at the time of crash. When a transaction is committed or aborted, some additional actions (like removing its entry from the transaction table, etc.) are taken after writing the commit or abort log record. When these actions are completed, the end (or completion) log record containing the transaction id is appended in the log. Thus, this record notes that a particular transaction has been committed or aborted. A checkpoint is also maintained by writing a begin_checkpoint and end_checkpoint record in the log. The LSN of the begin_checkpoint record is written to a special file which is accessed during recovery to find the last checkpoint information. When an end_checkpoint record is written, the contents of both the transaction table and the dirty page table are appended to the end of the log. To reduce the cost of checkpointing, fuzzy checkpointing is used so that DBMS can continue to execute transactions during checkpointing. The contents of the modified cache buffers need not be flushed to disk during checkpoint. This is because the transaction table and dirty page table, which are already appended to log on the stable storage, contain the information needed for recovery. Note that if the system fails during checkpointing, the special file containing the LSN of begin_checkpoint record of the previous checkpoint will be used for recovery. When a system restarts after a crash, the ARIES recovers from the crash in three phases: ? Analysis phase: This phase starts from the begin_checkpoint record and proceeds till the end of the log. When the end_checkpoint record in the log is encountered, the transaction table and dirty page table are accessed which were written in the log during checkpointing. During analysis, these two tables are reconstructed as follows: o If an end log record for a transaction T i is encountered in the transaction table, its entry is deleted from that table. o If any other type of log record for a transaction T j is encountered, its entry is inserted into the transaction table (if not present) and the last LSN is modified accordingly. o If the log record specifying a change in page P is encountered, an entry is made for that page (if not already present) in the dirty page table and associated recLSN field is modified. At the end of analysis phase, the necessary information for redo and undo phase has been compiled in transaction table and dirty page table. ? Redo phase: The redo phase actually reapplies the updates from the log to the database. In ARIES, the redo operation is not applied to only the committed transactions, rather, a starting point from which the redo phase should start is determined first, and from this point the redo phase begins. The smallest LSN, which indicates the log position from where the redo phase needs to be started, is determined from the dirty page table. Before this point, the changes to dirty pages have already been reflected to the database on disk. ? Undo phase: This phase rolls back all transactions that were not committed at the time of failure. A backward scan of the log is performed and all the transactions in the undo-list are rolled back. A compensating log record for each undo action is written in log. After undo phase, the recovery process is finished and normal processing can be started again. Learn More IBM DB2, MS SQL Server, and Oracle 8 use Write-ahead logging scheme for recovery from a system crash. Out of these, IBM DB2 uses ARIES, while other use schemes that are more or less similar to ARIES. To understand how ARIES works, consider three transactions T 1 , T 2 , and T 3 , where T 1 is updating page A, T 2 is updating pages B and C, and T 3 is updating page D. The log records at the point of crash are shown in Figure 11.7(a). The transaction 297
table and dirty page table at the time of checkpoint and after the analysis phase are shown in Figures 11.7(b) and 11.7(c), respectively. The analysis phase starts from the begin_checkpoint record, which is at LSN 5 and continues until it reaches the end of the log. The transaction table and dirty page table are reconstructed during analysis phase as given below: ? When the log record 7 is encountered, the status of transaction T 2 is changed to 'committed' in the transaction table. ? When the log record 8 is encountered, new entry for the transaction T 3 is made in transaction table and an entry for page D is made in dirty page table. During redo phase, the smallest LSN is determined from the dirty page table, which is 1. Thus, redo phase will start from the log record 1. The LSNs 1, 3, 4, and 8 indicate the change in page A, B, C, and D, respectively. Since these LSNs are not less than the smallest LSN, the updates on these pages need to be reapplied from the log. During undo phase, the transaction table is scanned to determine the transactions that need to be rolled back. The transaction T 3 is the only transaction that was active at the point of crash. Thus, T 3 needs to be rolled back. The undo phase begins at log record 8 (for transaction T 3) and proceeds backward to undo all the updates performed by transaction T 3 . Since in our example, log record 8 is the only update operation performed by T 3 , it needs to be undone. Fig. 11.7 An example of ARIES recovery algorithm ARIES has many advantages which are given here. ? It is simple and flexible to implement. ? It can support concurrency control techniques that involve locks of finer granularity. It also provides a variety of optimization techniques to improve concurrency control. ? It uses a number of techniques to reduce logging overhead, overheads of checkpoints, and time taken for recovery. 11.7 RECOVERY FROM CATASTROPHIC FAILURES 298
The recovery techniques discussed so far are applied in case of non-catastrophic failures. The recovery manager should also be able to deal with catastrophic failures. Though, such failures are very rare, still some techniques must be developed to recover from such failures. The main technique used to handle such failures is database backup (also known as dump). In this technique, the entire contents of

SA

MATCHING BLOCK 311/319 47F417BA15858476660.pdf (D123781188)

the database and the log are copied periodically onto a cheap storage medium such as magnetic tapes.

When a database backup is taken, the following actions are performed. 1. The execution of the currently running transactions is suspended. 2. All the log records currently residing in the main memory are written to the stable storage. 3. All the modified buffer blocks are force-written to the disk. 4. The contents of the database are copied to the stable storage. 5. A [dump] record is written to the log on the stable storage. 6. The execution of suspended transactions is resumed. Note that all the steps except step 4 are similar to the steps used for checkpoints. Thus, the database backup and checkpointing are similar processes to an extent. Like fuzzy checkpoints, the technique called fuzzy dump is also used in practice. In fuzzy dump technique, the transactions are allowed to continue while the dump is in progress. Since the system log is much smaller than the database itself, the backup of the log records is taken more frequently than the full database backup. The backup of the system log helps in redoing the effect of

90% **MATCHING BLOCK 314/319** SA 47F417BA15858476660.pdf (D123781188)

all the transactions that have been executed since the last backup.

In case of disk failure the most recent dump of the database is loaded from the magnetic tapes to the disk. The system log is then used to bring the database to the most recent consistent state by reapplying all the operations since the last backup. In case of environmental disasters such as flood, earthquakes, etc., the backup of the database taken on the stable storage at a remote site is used to recover the lost data. The copy of the database is made generally by writing each block of stable storage over a computer network. This is called remote backup. The site at which the transaction processing is performed is known as primary site and the remote site at which the backup is taken is known as secondary

95% **MATCHING BLOCK 312/319** SA Advanced DBMS.pdf (D166063498)

site. The remote site must be physically separated from the primary

site so that any kind of natural disaster does not damage the remote site. In case of natural disasters when the primary site fails, the remote site takes over the processing. But before that the recovery is performed at the remote site using the most recent backup of the data (may be outdated) and the log records received from the primary site. Once the recovery has been performed, the remote site can start the processing of the transactions. Note that each time the updates are performed at the primary site; the updates must be propagated to the remote site so that the remote site remains synchronized with the primary site. This can be achieved by sending all the

80% **MATCHING BLOCK 313/319** SA Advanced DBMS.pdf (D166063498)

log records from the primary site to the remote site where the backup

is taken. SUMMARY 1. The main aim of recovery is to restore the database to the most recent consistent state which existed prior to the failure. The recovery manager component of DBMS is responsible for performing the recovery operations. 2. The recovery manager ensures that the two important properties of transactions, namely, atomicity and durability are preserved. 3. There are several types of failures that can occur in a system and stop the normal execution of the transaction such as logical error, system error, computer failure (system crash), disk failure, physical problems, and environmental disasters. 4. Whenever a transaction needs to update the database, the disk pages containing the data items to be modified are first cached (buffered) by the cache manager into the main memory and then modified in the memory before being written back to the disk. 5. A cache directory is maintained to keep track of all the data items present in the buffers. 6. The data which is modified in the cache can be copied back to the disk using either of the two strategies, namely, in-place updating and shadow paging. 299
7.

100%	MATCHING BLOCK 315/319	SA	47F417BA15858476660.pdf (D123781188)

Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a

modified page in the cache buffer can be written back to the database disk. 8. The recovery manager

85%	MATCHING BLOCK 317/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

maintains a system log of all modifications to the database and stores it on the stable storage. 9.

A log is a sequence of log records that contains essential data for each transaction which has updated the database. 10. In case of update operation, the log record contains the before and after images of the portion of the database which have been modified by the transaction. 11. Before making any changes to the database, it is necessary to force-write all log records on the stable storage. This is known as write-ahead logging (WAL). 12. The process in which multiple log records are first collected in the log buffer and then copied to the stable storage in a single write operation is called log record buffering. Writing the buffered log to disk is known as log force. 13. To recover from a failure, basically two operations, namely, undo and redo are applied with the help of the log on the last consistent state of the database. 14. The undo operation undoes (reverses or rollbacks) the changes made to the database by an uncommitted transaction and restores

100%	MATCHING BLOCK 316/319	SA	Advanced DBMS.pdf (D166063498)

the database to the consistent state that existed before the

start of transaction. 15. The redo operation redoes the changes of a committed transaction and restores the database to the consistent state it would be at the end of the transaction. The redo operation is required when the changes of a committed transaction are not or partially reflected to the database on disk. 16. The recovery techniques are categorized mainly into two types, namely log-based recovery techniques and shadow paging. 17. The log-based recovery techniques maintain transaction logs to keep track of transaction operations. The log- based recovery techniques are further classified into two types, namely, techniques based on deferred update and techniques based on immediate update. 18. In deferred update technique, the transaction is not allowed to update the database on disk until the transaction enters into the partially committed state. 19. In immediate update technique, as soon as a data item is modified in cache, the disk copy is immediately updated. 20.

100%	MATCHING BLOCK 318/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

To reduce the time to recover from a crash, the DBMS periodically force-write all the modified buffer pages during normal execution

using a process known as checkpointing. 21. The shadow paging technique does not require the use of a log as both the after image and the before image of the data item to be modified are maintained on the disk. 22. In case of recovery when concurrent transactions are executing, two lists, namely, active list and commit list are maintained by the recovery system. All the active transactions are entered in the active list and all the committed transactions since the last checkpoint are entered in the commit list. 23. Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is an example of recovery algorithm which is widely used in the database systems. It uses steal/no-force approach for writing the modified buffers back to the database on the disk. 24. In ARIES, each log record is assigned a unique id called the log sequence number (LSN) in monotonically increasing order. This id indicates the address of the log record on the disk. 25. ARIES

76%	MATCHING BLOCK 319/319	SA	MCA-Sem-II-Database Management System.pdf (D143652738)

is based on three principles including write-ahead logging, repeating history during redo,

and logging changes during undo. 26. When a system restarts after a crash, the ARIES recovers from the crash in three phases—the first phase is the analysis phase, second phase is the redo phase, and the last phase is the undo phase. 300

Ouriginal by Turnitin

Hit and source - focused comparison, Side by Side

Submitted text	As student entered the text in the submitted document.
Matching text	As the text appears in the source.

1/319	**SUBMITTED TEXT** 18 WORDS	**73%** **MATCHING TEXT** 18 WORDS
	The concept of data independence, which helps the user to change the schema at one level of	
	SA 248E1130_RDBMS.docx (D165247738)	

2/319	**SUBMITTED TEXT** 10 WORDS	**100%** **MATCHING TEXT** 10 WORDS
	database system without having to change the schema at the	
	SA 248E1130_RDBMS.docx (D165247738)	

3/319	**SUBMITTED TEXT** 20 WORDS	**66%** **MATCHING TEXT** 20 WORDS
	database management system. A Database Management System (DBMS) is an integrated set of programs used to create and maintain a database.	
	SA BOOK SIZE.docx (D50092775)	

4/319	**SUBMITTED TEXT** 23 WORDS	**67%** **MATCHING TEXT** 23 WORDS
	of a DBMS is to provide a convenient and effective method of defining, storing, retrieving, and manipulating the data contained in the database.	
	SA BOOK SIZE.docx (D50092775)	

5/319	**SUBMITTED TEXT** 14 WORDS	**84%** **MATCHING TEXT** 14 WORDS
	Same information may be duplicated in several files. For example, the name and	same information may be duplicated in several places (files). For example, the address and
	SA Advanced DBMS.pdf (D166063498)	

6/319	**SUBMITTED TEXT** 27 WORDS	**41%** **MATCHING TEXT** 27 WORDS
	The first generation includes the database systems based on hierarchical and network data model. The second generation includes the database systems based on relational data model, and the	The first generation of DBMSs is based on hierarchical and network data models. The second generation of DBMSs includes relational data models. The third generation is based on the Object-Oriented Data Model (OODM) and the
	SA Advanced DBMS.pdf (D166063498)	

7/319	SUBMITTED TEXT	14 WORDS	76% MATCHING TEXT	14 WORDS
	the database to the state which existed prior to the start of the		the database to the state that existed prior to the occurrence of the	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

8/319	SUBMITTED TEXT	10 WORDS	100% MATCHING TEXT	10 WORDS
	the prestigious Association of Computing Machinery Turing Award for his work.			

SA 248E1130_RDBMS.docx (D165247738)

9/319	SUBMITTED TEXT	22 WORDS	54% MATCHING TEXT	22 WORDS
	database. They also began to add object-relational support to their databases. In the late 1990s, with the advent of World Wide Web,			

SA 248E1130_RDBMS.docx (D165247738)

10/319	SUBMITTED TEXT	13 WORDS	87% MATCHING TEXT	13 WORDS
	computer-aided design systems, knowledge-base and expert systems that store data having complex data types.			

SA 248E1130_RDBMS.docx (D165247738)

11/319	SUBMITTED TEXT	13 WORDS	92% MATCHING TEXT	13 WORDS
	the database schema by executing a set of data definition statements in DDL.		the original database schema by executing a set of data definition statements in the DDL. 2.	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

12/319	SUBMITTED TEXT	12 WORDS	100% MATCHING TEXT	12 WORDS
	Database administrator (DBA) is a person who has central control over			

SA DBMS_AIML_FINAL.pdf (D111167082)

13/319	SUBMITTED TEXT	15 WORDS	66% MATCHING TEXT	15 WORDS
	with an abstract view of the data by hiding certain details of how data is			

SA MCA-Sem-II-Database Management System.pdf (D143652738)

14/319	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

data without worrying about where it is located or how it is actually stored.

SA Database System.pdf (D165747767)

15/319	SUBMITTED TEXT	10 WORDS	100%	MATCHING TEXT	10 WORDS

ANSI/SPARC (American 10 National Standards Institute/Standards Planning and Requirements Committee)

SA BOOK SIZE.docx (D50092775)

16/319	SUBMITTED TEXT	17 WORDS	65%	MATCHING TEXT	17 WORDS

logical level. It describes what data is stored in the database, the relationships among the data

SA MCA-Sem-II-Database Management System.pdf (D143652738)

17/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

part of the database for a particular group of users.

SA MCA-Sem-II-Database Management System.pdf (D143652738)

18/319	SUBMITTED TEXT	47 WORDS	41%	MATCHING TEXT	47 WORDS

three-schema architecture. The internal level has an internal schema, which describes the physical storage structure of the database. The conceptual level has a conceptual schema, which describes the structure of entire database. The external level has external schemas or user views, which describe the part of the database

SA DBMS_AIML_FINAL.pdf (D111167082)

19/319	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

may be aware of certain details of the physical organization of the data. 11

SA 248E1130_RDBMS.docx (D165247738)

20/319	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

three-schema architecture, each user group refers only to its own external

SA 47F417BA15858476660.pdf (D123781188)

21/319	SUBMITTED TEXT	27 WORDS	39%	MATCHING TEXT	27 WORDS

retrieval, the data extracted from the database must be presented according to the need of the user. This process of transforming the requests and results between various levels

SA 248E1130_RDBMS.docx (D165247738)

22/319	SUBMITTED TEXT	13 WORDS	87%	MATCHING TEXT	13 WORDS

to change the schema at one level of the database system without

SA BOOK SIZE.docx (D50092775)

23/319	SUBMITTED TEXT	36 WORDS	75%	MATCHING TEXT	36 WORDS

the schema at the other levels. Data independence is of two types, namely, logical data independence and physical data independence. ? Logical data independence: It is the ability to change the conceptual schema without affecting the external schemas

SA DBMS.docx (D68067493)

24/319	SUBMITTED TEXT	20 WORDS	60%	MATCHING TEXT	20 WORDS

is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be

SA 47F417BA15858476660.pdf (D123781188)

25/319	SUBMITTED TEXT	13 WORDS	85%	MATCHING TEXT	13 WORDS

the network model permits the modeling of many-to-many relationships in data. The

SA all in one.pdf (D109420358)

26/319	SUBMITTED TEXT	18 WORDS	94% MATCHING TEXT	18 WORDS
	data models, where every data item of a particular type must have the same set of attributes, the		data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

27/319	SUBMITTED TEXT	14 WORDS	76% MATCHING TEXT	14 WORDS
	individual data items of the same type to have different set of attributes.		individual data items of the same type may have different sets of attributes.	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

28/319	SUBMITTED TEXT	12 WORDS	78% MATCHING TEXT	12 WORDS
	object-oriented data model and object-relational data model. The object-oriented data model			

SA 47F417BA15858476660.pdf (D123781188)

29/319	SUBMITTED TEXT	11 WORDS	100% MATCHING TEXT	11 WORDS
	an environment that is both convenient and efficient to use		an environment that is both convenient and efficient to use	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

30/319	SUBMITTED TEXT	11 WORDS	100% MATCHING TEXT	11 WORDS
	the number of sites over which the database is distributed,		the number of sites over which the database is distributed.	

SA Advanced DBMS.pdf (D166063498)

31/319	SUBMITTED TEXT	13 WORDS	75% MATCHING TEXT	13 WORDS
	communicate with each other through various communication media such as high-speed networks or		communicate with one another through various communication media, such as high-speed private networks or	

SA Advanced DBMS.pdf (D166063498)

32/319	SUBMITTED TEXT	37 WORDS	50%	MATCHING TEXT	37 WORDS

The rest of the program is sent to the host language compiler. The object codes of both the DML commands and the rest of the program are linked and sent to the query evaluation engine for execution. o The

SA Database System.pdf (D165747767)

33/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

Submitted	Matching
the high-level conceptual schema onto the implementation data model of the	the high-level conceptual schema onto the implementation data model of the

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

34/319	SUBMITTED TEXT	13 WORDS	95%	MATCHING TEXT	13 WORDS

Submitted	Matching
from an abstract data model to the implementation of the database. In	from an abstract data model to the implementation of the database proceeds in

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

35/319	SUBMITTED TEXT	15 WORDS	93%	MATCHING TEXT	15 WORDS

homogeneous and heterogeneous. In homogeneous distributed database system, all sites have identical database management system software,

SA 248E1130_RDBMS.docx (D165247738)

36/319	SUBMITTED TEXT	18 WORDS	61%	MATCHING TEXT	18 WORDS

A Database Management System (DBMS) is an integrated set of programs used to create and maintain a database.

SA BOOK SIZE.docx (D50092775)

37/319	SUBMITTED TEXT	27 WORDS	41%	MATCHING TEXT	27 WORDS

Submitted	Matching
The first generation includes the database systems based on hierarchical and network data model. The second generation includes the database systems based on relational data model and the	The first generation of DBMSs is based on hierarchical and network data models. The second generation of DBMSs includes relational data models. The third generation is based on the Object-Oriented Data Model (OODM) and the

SA Advanced DBMS.pdf (D166063498)

38/319	SUBMITTED TEXT	23 WORDS	67%	MATCHING TEXT	23 WORDS

of a DBMS is to provide a convenient and effective method of defining, storing, retrieving, and manipulating the data contained in the database. 5.

SA BOOK SIZE.docx (D50092775)

39/319	SUBMITTED TEXT	15 WORDS	89%	MATCHING TEXT	15 WORDS

database administrator (DBA). Database administrator (DBA) is a person who has central control over

SA DBMS_AIML_FINAL.pdf (D111167082)

40/319	SUBMITTED TEXT	13 WORDS	87%	MATCHING TEXT	13 WORDS

to change the schema at one level of the database system without

SA BOOK SIZE.docx (D50092775)

41/319	SUBMITTED TEXT	19 WORDS	72%	MATCHING TEXT	19 WORDS

the schema at the other levels. Data independence is of two types, namely, logical data independence and physical data independence. 15.

SA DBMS.docx (D68067493)

42/319	SUBMITTED TEXT	16 WORDS	100%	MATCHING TEXT	16 WORDS

focuses on what data is required and how it should be organized rather than what operations

SA 47F417BA15858476660.pdf (D123781188)

43/319	SUBMITTED TEXT	23 WORDS	65%	MATCHING TEXT	23 WORDS

Peter Chen in 1976 as a way to unify the network and relational database views. The E-R model views the real world as

SA 47F417BA15858476660.pdf (D123781188)

44/319	SUBMITTED TEXT 14 WORDS	84% MATCHING TEXT 14 WORDS
	and derived attributes: In some cases, two or more attribute values are related	and Non-Derived Attribute – In some cases, two or more attribute values are related,

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

45/319	SUBMITTED TEXT 17 WORDS	82% MATCHING TEXT 17 WORDS
	The value of the attribute Age can be determined from the current date and the value of	the value of Age can be determined from the current date and the value of

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

46/319	SUBMITTED TEXT 13 WORDS	87% MATCHING TEXT 13 WORDS
	The primary key of a weak entity type is formed by the	The primary key of a weak entity set is formed by the

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

47/319	SUBMITTED TEXT 12 WORDS	87% MATCHING TEXT 12 WORDS
	Entity Types An entity type that does not have any key attribute	Entity Types • • An entity type that does not have a key attribute

W https://present5.com/database-theory-jason-fan-outline-basic/

48/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	partial key is a set of attributes that can uniquely identify	

SA DBMS.docx (D68067493)

49/319	SUBMITTED TEXT 20 WORDS	60% MATCHING TEXT 20 WORDS
	one-to-one-to-many (1 : 1 : M) or many-to-one-to-one (M : 1 : 1), one-to-many-to-many (1 : M : N)	

SA DBMS.docx (D68067493)

50/319	SUBMITTED TEXT	12 WORDS	76%	MATCHING TEXT	12 WORDS

data objects in E-R diagrams. Each modeling methodology uses its own notation.

SA 47F417BA15858476660.pdf (D123781188)

51/319	SUBMITTED TEXT	14 WORDS	89%	MATCHING TEXT	14 WORDS

types are described by a set of attributes that includes all the attributes of

types is described by a set of attributes that includes all the attributes of

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

52/319	SUBMITTED TEXT	120 WORDS	100%	MATCHING TEXT	120 WORDS

The E-R diagrams discussed so far represents the basic concepts of a database schema. However, some aspects of a database such as inheritance among various entity types cannot be expressed using the basic E-R model. These aspects can be expressed by enhancing the E-R model. The resulting diagrams are known as enhanced E-R or EER diagrams and the model is called EER model. The basic E-R model can represent the traditional database applications such as typical data processing application of an organization effectively. On the other hand, the EER model is used to represent the new and complex database applications such as telecommunications, Geographical Information Systems (GIS), etc. This section discusses the extended E-R features including specialization, generalization, and aggregation and their representation using EER diagrams. 2.3.1

SA BOOK SIZE.docx (D50092775)

53/319	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

an attribute-defined specialization, and the attribute is called the defining attribute

SA BOOK SIZE.docx (D50092775)

54/319	SUBMITTED TEXT	12 WORDS	87%	MATCHING TEXT	12 WORDS

belong to more than one lower-level entity type within a single

belong to more than one lower- level entity set within a single

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

55/319	SUBMITTED TEXT	15 WORDS	83%	MATCHING TEXT	15 WORDS

by writing the condition next to the line connecting the subclass to the specialization circle (

SA BOOK SIZE.docx (D50092775)

56/319	SUBMITTED TEXT	16 WORDS	77%	MATCHING TEXT	16 WORDS

higher-level entity must belong to at least one of the lower-level entity types in the specialization.

higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization.

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

57/319	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

each participation of an entity type E in a relationship type R,

each participation of an entity type E in a relationship type R.

W https://present5.com/database-theory-jason-fan-outline-basic/

58/319	SUBMITTED TEXT	15 WORDS	90%	MATCHING TEXT	15 WORDS

E must participate in at least min and at most max relationship instances in R.

E participates in at least min and at most max relationship instances in R.

W https://present5.com/database-theory-jason-fan-outline-basic/

59/319	SUBMITTED TEXT	21 WORDS	81%	MATCHING TEXT	21 WORDS

completeness constraint. It determines whether an entity in higher-level entity set must belong to at least one of the lower-level entity sets

SA DBMS_AIML_FINAL.pdf (D111167082)

60/319	SUBMITTED TEXT	16 WORDS	100%	MATCHING TEXT	16 WORDS

focuses on what data is required and how it should be organized rather than what operations

SA 47F417BA15858476660.pdf (D123781188)

61/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

views the real world as a set of basic objects (

SA Database System.pdf (D165747767)

62/319	SUBMITTED TEXT	17 WORDS	58%	MATCHING TEXT	17 WORDS

Submitted	Matching
an entity type that has a key attribute is called a strong entity type. 17. A weak entity	An entity set that has a primary key is termed a strong entity set. For a weak entity

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

63/319	SUBMITTED TEXT	16 WORDS	68%	MATCHING TEXT	16 WORDS

that has an independent existence in the real world. If an entity has a physical existence,

SA Database System.pdf (D165747767)

64/319	SUBMITTED TEXT	13 WORDS	88%	MATCHING TEXT	13 WORDS

some other aspects of a database such as inheritance among various entity types. 25.

SA BOOK SIZE.docx (D50092775)

65/319	SUBMITTED TEXT	30 WORDS	68%	MATCHING TEXT	30 WORDS

or relation) r of the relation schema R(A 1 , A 2 , ..., A n) is a set of n-tuples t.

SA BOOK SIZE.docx (D50092775)

66/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

the number of tuples in a relation is known as cardinality.

SA BOOK SIZE.docx (D50092775)

67/319	SUBMITTED TEXT 39 WORDS	56% MATCHING TEXT 39 WORDS
	Ordering of Tuples in a Relation: Since a relation is a set of tuples and a set has no particular order among its elements, tuples in a relation do not have any specified order. However, tuples in a relation	

SA DBMS_AIML_FINAL.pdf (D111167082)

68/319	SUBMITTED TEXT 10 WORDS	100% MATCHING TEXT 10 WORDS
	E-R database schema by a collection of relation schemas. 68	E-R database schema by a collection of relation schemas.

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

69/319	SUBMITTED TEXT 48 WORDS	60% MATCHING TEXT 48 WORDS
	relation schema R with one attribute for each member of the set as {a 1 , a 2 , ..., a m } ∪ {A 1 , A 2 , ..., A n } The primary key	relation schema called R with one attribute for each member of the set: { a 1 , a 2 , . . . , a m } ∪ {b 1 , b 2 , . . . , b n } The primary key

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

70/319	SUBMITTED TEXT 16 WORDS	80% MATCHING TEXT 16 WORDS
	foreign key in a referencing relation that matches a primary key in a referenced relation	

SA DBMS_AIML_FINAL.pdf (D111167082)

71/319	SUBMITTED TEXT 22 WORDS	47% MATCHING TEXT 22 WORDS
	two entity types, namely, AUTHOR with the primary key A_ID, and BOOK with the primary key ISBN. Since the relationship type has	two entity sets: • instructor with the primary key ID. • student with the primary key ID. Since the relationship set has

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

72/319	SUBMITTED TEXT 16 WORDS	57% MATCHING TEXT 16 WORDS
	one-to-many or many-to-one Primary key of entity type on the 'many' side of the relationship	one-to-many relationship set, the primary key of entity set on the "many" side of the relationship

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

73/319	SUBMITTED TEXT 37 WORDS	64% MATCHING TEXT 37 WORDS
	n-ary without arrow on its edges Union of primary key attributes from the participating entity types n-ary with arrow on one of its edges Primary key of the entity type not on the 'arrow' side of relationship	n-ary relationship set without any arrows on its edges, the union of the primary key- attributes from the participating entity sets becomes the primary key. ❖ For an n-ary relationship set with an arrow on one of its edges, the primary keys of the entity sets not on the "arrow" side of the relationship

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

74/319	SUBMITTED TEXT 21 WORDS	54% MATCHING TEXT 21 WORDS
	weak entity type as the primary key of the weak entity type refers to the primary key of the strong entity	

SA BOOK SIZE.docx (D50092775)

75/319	SUBMITTED TEXT 21 WORDS	100% MATCHING TEXT 21 WORDS
	A003 002-678-980-4 5 A003 004-765-409-5 4 A004 003-456-533-8 9 A005 002-678-980-4 7 A006 001-354-921-1 7 A006 002-678-880-2 4 A007 004-765-359-3 3 A008 001-987-760-9 7 A009 001-987-650-5 8 A010 003-456-433-6 5	A a1 a1 a1 a2 a2 a2 a3 a3 a3

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

76/319	SUBMITTED TEXT 20 WORDS	50% MATCHING TEXT 20 WORDS
	the relationship type is many-to-many, the primary key for the REVIEW relation is the union of the primary key of	

SA BOOK SIZE.docx (D50092775)

77/319	SUBMITTED TEXT 26 WORDS	65% MATCHING TEXT 26 WORDS
	key of the entity type, the primary key of new relation will be the combination of the primary key and the multivalued attribute of the entity	

SA DBMS_AIML_FINAL.pdf (D111167082)

78/319	SUBMITTED TEXT 16 WORDS	70% MATCHING TEXT 16 WORDS
	the primary key for the relationship type is the union of the primary keys of	the primary key for the relationship R can be constructed by the union of the primary keys of

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

79/319	SUBMITTED TEXT 16 WORDS	65% MATCHING TEXT 16 WORDS
	the primary key for the relationship type WRITES is the union of the primary keys of	the primary key for the relationship R can be constructed by the union of the primary keys of

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

80/319	SUBMITTED TEXT 29 WORDS	81% MATCHING TEXT 29 WORDS
	relational algebra and relational calculus. 4.1 RELATIONAL ALGEBRA Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input and	relational calculus and the domain relational calculus are RELATIONAL ALGEBRA The relational algebra is a procedural query language. consists of a set of operations that take one or two relations as input and

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

81/319	SUBMITTED TEXT 20 WORDS	57% MATCHING TEXT 20 WORDS
	the primary key of the relation includes the primary key of the entity type and the primary key of the	

SA DBMS_AIML_FINAL.pdf (D111167082)

82/319	SUBMITTED TEXT 23 WORDS	47% MATCHING TEXT 23 WORDS
	The primary key of the relation, say COPYRIGHT, includes the primary key of the entity type PUBLISHER and the primary key of the	

SA DBMS_AIML_FINAL.pdf (D111167082)

83/319	SUBMITTED TEXT 21 WORDS	64% MATCHING TEXT 21 WORDS
	aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as	

SA BOOK SIZE.docx (D50092775)

84/319	SUBMITTED TEXT	10 WORDS	100%	MATCHING TEXT	10 WORDS

Tuple Relational Calculus The tuple relational calculus is based on

SA BOOK SIZE.docx (D50092775)

85/319	SUBMITTED TEXT	13 WORDS	80%	MATCHING TEXT	13 WORDS

A simple tuple relational calculus query is of the form {T|P(T)} where, T

SA BOOK SIZE.docx (D50092775)

86/319	SUBMITTED TEXT	16 WORDS	90%	MATCHING TEXT	16 WORDS

is the set of all tuples T for which the condition P(T) evaluates to true.

SA DCAP 011.docx (D142453284)

87/319	SUBMITTED TEXT	34 WORDS	61%	MATCHING TEXT	34 WORDS

by using these rules 97 1. An atom is a formula. 2. If F is a formula, then so is ¬F. 3. If F 1 and F 2 are formulae, then so are

by using the following An atom is a formula. • If P 1 is a formula, then so are ¬p 1 and (P 1) If P 1 and P 2 are formulae, then so are

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

88/319	SUBMITTED TEXT	17 WORDS	87%	MATCHING TEXT	17 WORDS

Each of the atoms evaluates to either true or false for a specific combination of tuples

SA BOOK SIZE.docx (D50092775)

89/319	SUBMITTED TEXT	14 WORDS	96%	MATCHING TEXT	14 WORDS

it appears in (∃T) or (∀T) clause, otherwise, it is free. A tuple variable

SA BOOK SIZE.docx (D50092775)

90/319	SUBMITTED TEXT	62 WORDS	71%	MATCHING TEXT	62 WORDS

in a formula F, which is an atom. ? A tuple variable T is free or bound in a formula of the forms (F 1 ∧ F 2), (F 1 ∨ F 2) and ¬F depending on whether it is free or bound in F 1 or F 2 (

SA BOOK SIZE.docx (D50092775)

91/319	SUBMITTED TEXT	40 WORDS	91%	MATCHING TEXT	40 WORDS

is free in F. ? All free occurrences of a tuple variable T in F are bound in a formula F′ of the form F′ = (∃T) (F) or F′ = (∀T)(F). The tuple variable is bound to the quantifier specified in F′.

SA BOOK SIZE.docx (D50092775)

92/319	SUBMITTED TEXT	15 WORDS	85%	MATCHING TEXT	15 WORDS

one) tuple assigned to free occurrences of T in F, otherwise it is false. 7. (∀

SA BOOK SIZE.docx (D50092775)

93/319	SUBMITTED TEXT	21 WORDS	66%	MATCHING TEXT	21 WORDS

true for every tuple (universal) assigned to free occurrences of T in F, otherwise it is false. Transforming the Universal and

SA BOOK SIZE.docx (D50092775)

94/319	SUBMITTED TEXT	19 WORDS	73%	MATCHING TEXT	19 WORDS

more precisely by introducing the concept of the domain of a tuple relational calculus expression, E. The domain of

SA BOOK SIZE.docx (D50092775)

95/319	SUBMITTED TEXT	21 WORDS	73%	MATCHING TEXT	21 WORDS

is the set of all values appearing either as constant values in the expression E, or appearing in any of the

SA BOOK SIZE.docx (D50092775)

96/319	SUBMITTED TEXT 15 WORDS	70% MATCHING TEXT 15 WORDS
	is the set of all values that appear as constant in E, or appear in	

SA BOOK SIZE.docx (D50092775)

97/319	SUBMITTED TEXT 47 WORDS	91% MATCHING TEXT 47 WORDS
	x 1 , x 2 , ..., x n), where R is a relation with n attributes and x 1 , x 2 , ..., x n are domain variables or domain constants	x 1, x 2) , ...x n &lt; Єr, where r is a relation on n attributes and x 1, x 2, ...x n are domain variables or domain constants. •

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

98/319	SUBMITTED TEXT 60 WORDS	67% MATCHING TEXT 60 WORDS
	x is a domain variable and c is a constant in the domain of the attribute for which x is a domain variable. Formulas are built from atoms by using these rules 1. An atom is a formula. 2. If F is a formula, then so is ¬F. 3. If F 1 and F 2 are formulae, then so are	x is a domain variable, Θ is a comparison operator, and c is a constant in the domain of the attribute for which x is a domain variable. build up formulae from atoms by using the following An atom is a formula. • If P 1 is a formula, then so are ¬p 1 and (P 1) If P 1 and P 2 are formulae, then so are

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

99/319	SUBMITTED TEXT 61 WORDS	88% MATCHING TEXT 61 WORDS
	a 4 , a 6 , a 7 | (∃a 2) (AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 , a 6 , a 7) ∧ a 2 ="	A a1 a1 a1 a2 a2 a2 a3 a3 a3 a2 a2

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

100/319	SUBMITTED TEXT 15 WORDS	70% MATCHING TEXT 15 WORDS
	A simple domain relational calculus query is of the form {T | P(T)} where, T	

SA BOOK SIZE.docx (D50092775)

101/319	SUBMITTED TEXT 62 WORDS	88% MATCHING TEXT 62 WORDS
	b 2 , b 4 | (∃b 3) (∃b 7) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧	B b1 b2 b3 B b1 b2 b1 b1 b2 b1 b2 (

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

102/319	SUBMITTED TEXT 62 WORDS	88% MATCHING TEXT 62 WORDS
	b 1 , b 2 , b 4 | (∃b 3) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ (	B b1 b2 b3 B b1 b2 b1 b1 b2 b1 b2 (

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

103/319	SUBMITTED TEXT 83 WORDS	70% MATCHING TEXT 83 WORDS
	a 1 , a 2 , a 7 , b 3 | (∃c 1) (∃b 1) (∃c 2) (∃b 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ 2 ="	a4 a5 a5 Q B b1 b2 b3 B b1 b2 b1 b1 b2 b1

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

104/319	SUBMITTED TEXT 13 WORDS	87% MATCHING TEXT 13 WORDS
	atoms evaluate to either true or false for a specific set of	

SA BOOK SIZE.docx (D50092775)

105/319	SUBMITTED TEXT 121 WORDS	44% MATCHING TEXT 121 WORDS
	b 2 , b 4 , a 2 , a 7 | (∃ a1) (∃c 1) (∃c 2) (∃b 1) (∃b 8) (∃p 1) (∃p 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8 ∧ AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 ,	b 1 b 1 b 1 a 1 a 1 b 2 a 5 a 2 a 2 b 1 a 3 a 3 b 1 a 2 a 4 b 2 a 5 b 1 a 5 b 2 B A B A b 1 a 1 b 2 a 2 b 3 a 3 a 4 a 5

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

106/319	SUBMITTED TEXT 132 WORDS	45% MATCHING TEXT 132 WORDS
	c 2) (∃c 1) (∃a 1) (∃a 2) (BOOK(b 1 , b 2 , b 3 , b 4 , b 5 , b 6 , b 7 , b 8) ∧ AUTHOR(a 1 , a 2 , a 3 , a 4 , a 5 , a 6 , a 7) ∧ AUTHOR_BOOK(c 1 , c 2) ∧ b 1 = c 2 ∧ c 1 = a 1 ∧ a 2 ="	

SA Unit-4 (Part-II).docx (D76435895)

107/319	SUBMITTED TEXT 23 WORDS	93% MATCHING TEXT 23 WORDS
	Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input, and	relational algebra is a procedural query language. consists of a set of operations that take one or two relations as input and

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

108/319	SUBMITTED TEXT 13 WORDS	87% MATCHING TEXT 13 WORDS
	The expressive power of relational algebra is frequently used as a metric	

SA MCSDSC-2.2 Relational database Management system.pdf (D151484310)

109/319	SUBMITTED TEXT 19 WORDS	69% MATCHING TEXT 19 WORDS
	There are three types of outer joins, namely, left outer join, right outer join, and full outer join. 15.	There are actually three forms of the outer-join left outer join (), right outer join () and full outer join ().

W http://www.iete-elan.ac.in/SolQP/AC14.htm

110/319	SUBMITTED TEXT 18 WORDS	66% MATCHING TEXT 18 WORDS
	to determine whether a relation schema is in desirable normal form, information about the real world entity that	To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

111/319	SUBMITTED TEXT 21 WORDS	64% MATCHING TEXT 21 WORDS
	aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as	

SA BOOK SIZE.docx (D50092775)

112/319	SUBMITTED TEXT 11 WORDS	95% MATCHING TEXT 11 WORDS
	each relation schema contains a subset of the attributes of R.	Each relation Schema Ri contains a subset of the attributes of R. –

W https://present5.com/database-theory-jason-fan-outline-basic/

113/319	SUBMITTED TEXT 13 WORDS	87% MATCHING TEXT 13 WORDS
	The expressive power of relational algebra is frequently used as a metric	

SA MCSDSC-2.2 Relational database Management system.pdf (D151484310)

114/319	SUBMITTED TEXT	36 WORDS	64%	MATCHING TEXT	36 WORDS

that each attribute in R must appear in at least one relation schema R i so that no attributes are lost. Formally, ∪ R i = R This is called attribute preservation property of decomposition.

SA BOOK SIZE.docx (D50092775)

115/319	SUBMITTED TEXT	43 WORDS	100%	MATCHING TEXT	43 WORDS

for any two tuples t 1 and t 2 in r that have t 1 [X] = t 2 [X], they must also have t 1 [Y] = t 2 [Y].

SA Sem III_ BCA_B21CA05DC.docx (D165628090)

116/319	SUBMITTED TEXT	13 WORDS	83%	MATCHING TEXT	13 WORDS

whenever two tuples of r agree on their X value, they must

SA MCA-Sem-II-Database Management System.pdf (D143652738)

117/319	SUBMITTED TEXT	15 WORDS	76%	MATCHING TEXT	15 WORDS

Closure of a Set of Functional Dependencies For a given set F of functional dependencies,	Closure of a Set of Functional Dependencies: Given a set F set functional dependencies,

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

118/319	SUBMITTED TEXT	22 WORDS	52%	MATCHING TEXT	22 WORDS

a trivial dependency if it is satisfied by all relations. For example, $X \rightarrow X$ is satisfied by all relations having attribute

SA BOOK SIZE.docx (D50092775)

119/319	SUBMITTED TEXT	25 WORDS	87%	MATCHING TEXT	25 WORDS

The set of all FDs that are implied by a given set F of FDs is called the closure of F, denoted as F+.

SA MCSDSC-2.2 Relational database Management system.pdf (D151484310)

120/319	SUBMITTED TEXT	37 WORDS	46%	MATCHING TEXT	37 WORDS

B ⊆ A, then A → B holds. ? Augmentation rule: If A → B holds, then AC → BC holds. ? Transitivity rule: If both A → B and B → C hold, then A → C holds.

SA 248E1130_RDBMS.docx (D165247738)

121/319	SUBMITTED TEXT	48 WORDS	43%	MATCHING TEXT	48 WORDS

rule: If A → BC then A → B holds and A → C holds. ? Union rule: If A → B and A → C hold, then A → BC holds. ? Pseudotransitivity rule: If A → B holds and BC → D holds, then AC → D holds.

SA 248E1130_RDBMS.docx (D165247738)

122/319	SUBMITTED TEXT	19 WORDS	79%	MATCHING TEXT	19 WORDS

schema R(A, B, C, D, E, G) and the set F of functional dependencies { A → B,

SA 248E1130_RDBMS.docx (D165247738)

123/319	SUBMITTED TEXT	19 WORDS	83%	MATCHING TEXT	19 WORDS

schema R(A, B, C, D, E, G, H) and the set F of functional dependencies {A → B,

SA 248E1130_RDBMS.docx (D165247738)

124/319	SUBMITTED TEXT	20 WORDS	70%	MATCHING TEXT	20 WORDS

given set, any database that satisfies the simplified set of FDs will also satisfy the given set and vice versa.

SA 248E1130_RDBMS.docx (D165247738)

125/319	SUBMITTED TEXT	15 WORDS	87%	MATCHING TEXT	15 WORDS
	in F c is unique, that means, there are no two dependencies			in F c is unique. That is, there are no two dependencies	

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

126/319	SUBMITTED TEXT	47 WORDS	52%	MATCHING TEXT	47 WORDS

Attribute A is extraneous in X if A ∈ X, and F logically implies (F−{X → Y}) ∪ {(X−A) → Y}. ? Attribute A is extraneous in Y if A ∈ Y, and the set of FDs (F−{X → Y}) ∪ {X → (Y−A)} logically implies F. 153

SA 248E1130_RDBMS.docx (D165247738)

127/319	SUBMITTED TEXT	13 WORDS	95%	MATCHING TEXT	13 WORDS

A canonical cover for a set of functional dependencies can be computed

SA 248E1130_RDBMS.docx (D165247738)

128/319	SUBMITTED TEXT	18 WORDS	88%	MATCHING TEXT	18 WORDS

Normal forms 154 A relation is said to be in a particular normal form if it satisfies certain

SA DCAP 011.docx (D142453284)

129/319	SUBMITTED TEXT	33 WORDS	58%	MATCHING TEXT	33 WORDS

The normal forms are used to ensure that various types of anomalies and inconsistencies are removed from the database. Initially, Codd proposed three normal forms namely, first normal form (1NF), second normal form (2NF),

SA DBMS Block 2.pdf (D149011992)

130/319	SUBMITTED TEXT	17 WORDS	97%	MATCHING TEXT	17 WORDS
	Second Normal Form (2NF) The second normal form is based on the concept of full functional dependency.			Second Normal Form (2NF) The Second Normal Form (2NF) is based on the concept of full functional dependency,	

SA Advanced DBMS.pdf (D166063498)

131/319	SUBMITTED TEXT	15 WORDS	96%	MATCHING TEXT	15 WORDS

R is said to be in first normal form (1NF) if and only if

SA DCAP 011.docx (D142453284)

132/319	SUBMITTED TEXT 17 WORDS	91% MATCHING TEXT 17 WORDS
	in second normal form (2NF) if every non-key attribute A in R is fully functionally dependent on	in second normal form (2 NF) if every non- attribute A in R is fully functionally dependent on

W https://present5.com/database-theory-jason-fan-outline-basic/

133/319	SUBMITTED TEXT 13 WORDS	84% MATCHING TEXT 13 WORDS
	Normal Form (3NF) The third normal form is based on the concept of	Normal Form (2NF) Second Normal Form (2NF) is based on the concept of

SA Advanced DBMS.pdf (D166063498)

134/319	SUBMITTED TEXT 12 WORDS	76% MATCHING TEXT 12 WORDS
	prime attribute, if it is not a part of candidate key of R.	

SA Database System.pdf (D165747767)

135/319	SUBMITTED TEXT 18 WORDS	82% MATCHING TEXT 18 WORDS
	that is neither a candidate key nor a subset of any key of R and both X →	

SA BOOK SIZE.docx (D50092775)

136/319	SUBMITTED TEXT 20 WORDS	86% MATCHING TEXT 20 WORDS
	A relation schema R is in third normal form (3NF) with a set F of functional dependencies if, for	

SA MCA-Sem-II-Database Management System.pdf (D143652738)

137/319	SUBMITTED TEXT 23 WORDS	70% MATCHING TEXT 23 WORDS
	A relation schema R is in Boyce-Codd normal form (BCNF) if, for every FD X → A in F, where X is	A relation schema R is in (Boyce-Codd Normal Form) BCNF if whenever a FD X A in R, then X is

W https://present5.com/database-theory-jason-fan-outline-basic/

138/319	SUBMITTED TEXT 27 WORDS	48% MATCHING TEXT 27 WORDS
	R, at least one of the following statements hold ? X → Y is a trivial FD, that is, Y ⊆ X. ? X is a superkey for R. ?	

SA 248E1130_RDBMS.docx (D165247738)

139/319	SUBMITTED TEXT 24 WORDS	84% MATCHING TEXT 24 WORDS
	R is in third normal form (3NF) if and only if it satisfies 2NF and every nonkey attribute is non-transitively dependent on the primary key.	

SA Unit-4 (Part-II).docx (D76435895)

140/319	SUBMITTED TEXT 64 WORDS	85% MATCHING TEXT 64 WORDS
	decomposition of R into two relations R 1 and R 2 form a lossless decomposition if at least one of the following dependencies is in F + . R 1 ∩ R 2 → R 1 – R 2 R 1 ∩ R 2 →	decomposition of R into R 1 and R 2 a lossless decomposition, if at least one of the following dependencies is in F + : R 1 ∩ R 2 → R 1 R 1 ∩ R 2 → R 2

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

141/319	SUBMITTED TEXT 16 WORDS	90% MATCHING TEXT 16 WORDS
	that no spurious tuples are obtained when a natural join operation is applied to the relations	

SA BOOK SIZE.docx (D50092775)

142/319	SUBMITTED TEXT 12 WORDS	100% MATCHING TEXT 12 WORDS
	it is a necessary condition only if all constraints are functional dependencies. 162	

SA 248E1130_RDBMS.docx (D165247738)

143/319	SUBMITTED TEXT 37 WORDS	64% MATCHING TEXT 37 WORDS
	of FDs into relations R 1 , R 2 , ..., R n is dependency preserving if F' + = F + , where F'= F	of R into R 1 , R 2 , ... R n is a dependency preserving decomposition, if either F' = F or F '+ = F +

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

144/319	SUBMITTED TEXT 34 WORDS	38% MATCHING TEXT 34 WORDS
	decomposition of R, the projection of F to R i is the set F i of all FDs in F + (not just F) that include only attributes of	

SA 248E1130_RDBMS.docx (D165247738)

Ouriginal
by Turnitin

145/319	SUBMITTED TEXT	20 WORDS	80%	MATCHING TEXT	20 WORDS

is satisfied by all relations on schema R, then X →→ Y is a trivial dependency on schema R,

SA 248E1130_RDBMS.docx (D165247738)

146/319	SUBMITTED TEXT	23 WORDS	91%	MATCHING TEXT	23 WORDS

A relation schema R is in fourth normal form (4NF) with respect to a set F of functional and multivalued dependencies if, for

SA 248E1130_RDBMS.docx (D165247738)

147/319	SUBMITTED TEXT	41 WORDS	89%	MATCHING TEXT	41 WORDS

R 1 = (X ∪ Y) and R2 = (R–Y) based on MVD X →→ Y that holds in R, the decomposition has the lossless-join property. This condition is necessary and sufficient for decomposing a schema into two schemas that

SA BOOK SIZE.docx (D50092775)

148/319	SUBMITTED TEXT	13 WORDS	87%	MATCHING TEXT	13 WORDS

with respect to a set F of functional and multivalued dependencies if

with respect to a set D of functional and multivalued dependencies if,

W http://www.iete-elan.ac.in/SolQP/AC14.htm

149/319	SUBMITTED TEXT	17 WORDS	95%	MATCHING TEXT	17 WORDS

The relation schemas R 1 and R 2 form lossless-join decomposition of R

SA BOOK SIZE.docx (D50092775)

150/319	SUBMITTED TEXT	33 WORDS	76%	MATCHING TEXT	33 WORDS

Let R be a relation schema and R 1 , R 2 , ..., R n be the decomposition of R, R is said to satisfy

SA MCSDSC-2.2 Relational database Management system.pdf (D151484310)

151/319 **SUBMITTED TEXT** 59 WORDS **85%** **MATCHING TEXT** 59 WORDS

A relation schema R is in fifth normal form (5NF) with respect to a set F of functional, multivalued, and join dependencies if, for every non-trivial join dependency *(R 1 , R 2 , ..., R n) in F + , every R i is a superkey of R.

SA BOOK SIZE.docx (D50092775)

152/319 **SUBMITTED TEXT** 30 WORDS **60%** **MATCHING TEXT** 30 WORDS

that each attribute in R must appear in at least one relation schema R i so that no attributes are lost. This is called attribute preservation property of decomposition. 6.

SA BOOK SIZE.docx (D50092775)

153/319 **SUBMITTED TEXT** 27 WORDS **84%** **MATCHING TEXT** 27 WORDS

The set of all FDs that are implied by a given set of FDs is called the closure of F, denoted as F + . 8.

SA MCSDSC-2.2 Relational database Management system.pdf (D151484310)

154/319 **SUBMITTED TEXT** 16 WORDS **86%** **MATCHING TEXT** 16 WORDS

A relation is said to be in a particular normal form if it satisfies certain

SA DCAP 011.docx (D142453284)

155/319 **SUBMITTED TEXT** 17 WORDS **100%** **MATCHING TEXT** 17 WORDS

first normal form (1NF), second normal form (2NF), third normal form (3NF), and Boyce-Codd normal form (BCNF)

SA DBMS.docx (D142535242)

156/319 **SUBMITTED TEXT** 14 WORDS **96%** **MATCHING TEXT** 14 WORDS

Submitted text	Matching text
The second normal form is based on the concept of full functional dependency. 17.	The Second Normal Form (2NF) is based on the concept of full functional dependency,

SA Advanced DBMS.pdf (D166063498)

157/319	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

A relation is said to be in first normal form (1NF) if

SA DBMS Block 2.pdf (D149011992)

158/319	SUBMITTED TEXT	12 WORDS	95%	MATCHING TEXT	12 WORDS

third normal form is based on the concept of transitive dependency. 18.

SA DBMS.docx (D68067493)

159/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

can be placed anywhere in the file, where there is

SA U234.pdf (D109498494)

160/319	SUBMITTED TEXT	19 WORDS	89%	MATCHING TEXT	19 WORDS

the number of possible values a hash field can take is much larger than the number of available addresses

SA MCA-Sem-II-Database Management System.pdf (D143652738)

161/319	SUBMITTED TEXT	16 WORDS	71%	MATCHING TEXT	16 WORDS

collision, occurs when hash field value of a record to be inserted hashes to an address

SA MCA-Sem-II-Database Management System.pdf (D143652738)

162/319	SUBMITTED TEXT	12 WORDS	95%	MATCHING TEXT	12 WORDS

Submitted	Matching
a collection of buckets, with one primary page and additional overflow pages.	a collection of buckets, with one primary page and possibly additional overflow pages

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

163/319	SUBMITTED TEXT	32 WORDS	71%	MATCHING TEXT	32 WORDS

overflow space. In addition, a pointer field is associated with each record location. A collision is resolved by placing the new record in overflow space and the pointer field of occupied hash address

SA MCA-Sem-II-Database Management System.pdf (D143652738)

164/319	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

requires its own algorithms for insertion, retrieval, and deletion of records.

SA MCA-Sem-II-Database Management System.pdf (D143652738)

165/319	SUBMITTED TEXT	15 WORDS	96%	MATCHING TEXT	15 WORDS

the hash function to be modified dynamically to accommodate the growth or shrinkage of database

SA 248E1130_RDBMS.docx (D165247738)

166/319	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

the number of bits on which the bucket contents are based,

SA MCA-Sem-II-Database Management System.pdf (D143652738)

167/319	SUBMITTED TEXT	22 WORDS	43%	MATCHING TEXT	22 WORDS

pointer points to the first record with that clustering attribute value. Other records with the same clustering attribute value are stored sequentially after

SA 248E1130_RDBMS.docx (D165247738)

168/319	SUBMITTED TEXT	10 WORDS	100%	MATCHING TEXT	10 WORDS

is less than or equal to the search-key value.

SA 248E1130_RDBMS.docx (D165247738)

169/319	SUBMITTED TEXT	15 WORDS	73%	MATCHING TEXT	15 WORDS

index, just as any other sequential file and construct a primary index on the index

SA 248E1130_RDBMS.docx (D165247738)

170/319	SUBMITTED TEXT	17 WORDS	64%	MATCHING TEXT	17 WORDS

to search a record, we first apply binary search on the second level index to find the

SA 248E1130_RDBMS.docx (D165247738)

171/319	SUBMITTED TEXT	24 WORDS	48%	MATCHING TEXT	24 WORDS

less than or equal to the search-key. The pointer corresponding to this value points to the block of the first level index that contains

SA 248E1130_RDBMS.docx (D165247738)

172/319	SUBMITTED TEXT	24 WORDS	48%	MATCHING TEXT	24 WORDS

less than or equal to the search-key. The pointer corresponding to this value directs us to the block of file that contains the required record.

SA 248E1130_RDBMS.docx (D165247738)

173/319	SUBMITTED TEXT	18 WORDS	63%	MATCHING TEXT	18 WORDS

is split into two nodes at the same level and middle value is transferred to its parent node.

SA MCA-Sem-II-Database Management System.pdf (D143652738)

174/319	SUBMITTED TEXT	20 WORDS	76%	MATCHING TEXT	20 WORDS

more entries can be fit into an internal node of a B + - tree than corresponding B-tree.

SA MCA-Sem-II-Database Management System.pdf (D143652738)

175/319	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS
A collection of operations that form a single logical unit of work				a collection of operations that form a single logical unit of work	

SA Advanced DBMS.pdf (D166063498)

176/319	SUBMITTED TEXT	34 WORDS	89%	MATCHING TEXT	34 WORDS
read(A); A:=A–100; write(A); read(B); B:= B+100; write(B); If transaction				read(A) A := A – 50 write(A) read(B) B := B + 50 write(B) TRANSACTIONS	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

177/319	SUBMITTED TEXT 29 WORDS	55% MATCHING TEXT 29 WORDS
	The execution of transaction T 1 should also preserve the consistency of the database, that is, the sum of the values of account A and B should be $3500	the execution of the funds transfer preserve the consistency of the database. That is, the value of the sum of the balances of A and B must be

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

178/319	SUBMITTED TEXT 15 WORDS	96% MATCHING TEXT 15 WORDS
	the hash function to be modified dynamically to accommodate the growth or shrinkage of database	

SA 248E1130_RDBMS.docx (D165247738)

179/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	Whenever a transaction is submitted to a DBMS for execution,	Whenever a transaction is submitted to a DBMS for execution,

W https://present5.com/database-theory-jason-fan-outline-basic/

180/319	SUBMITTED TEXT 57 WORDS	43% MATCHING TEXT 57 WORDS
	by ensuring that either ? all the changes made by the transaction are written to the disk before the transaction completes, or ? the information about all the changes made by the transaction is written to the disk and is sufficient to enable the database system to reconstruct the changes when the database system is restarted after the failure. 9.2	

SA DBMS_AIML_FINAL.pdf (D111167082)

181/319	SUBMITTED TEXT 21 WORDS	77% MATCHING TEXT 21 WORDS
	executed the read operation. The WRITE operation transfers the data item from the local buffer of the transaction back to the database.	

SA DBMS_AIML_FINAL.pdf (D111167082)

182/319	SUBMITTED TEXT	17 WORDS	97%	MATCHING TEXT	17 WORDS

The only way to undo the effects of a committed transaction is to execute a compensating transaction,

SA DBMS_AIML_FINAL.pdf (D111167082)

183/319	SUBMITTED TEXT	16 WORDS	94%	MATCHING TEXT	16 WORDS

increases the throughput of the system (the number of transactions executed in a given amount of time).

SA DCAP 011.docx (D142453284)

184/319	SUBMITTED TEXT	14 WORDS	90%	MATCHING TEXT	14 WORDS
transactions and also preserve the order in which the instructions appear in each individual transaction.				transactions and must preserve the order in which the instructions appear in each individual transaction.	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

185/319	SUBMITTED TEXT	20 WORDS	57%	MATCHING TEXT	20 WORDS

short transaction may have to wait for a long time, leading to unpredictable delays in executing a transaction. On the

SA DCAP 011.docx (D142453284)

186/319	SUBMITTED TEXT	38 WORDS	54%	MATCHING TEXT	38 WORDS

Consider two transactions T 1 and T 5 , where T 1 transfers $100 from account A to account B and T 5 transfers $50 from account B to account

SA BOOK SIZE.docx (D50092775)

187/319	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

r, w, c, and a, for the operations read, write, commit, and abort, respectively.

SA DBMS Block 2.pdf (D149011992)

188/319	SUBMITTED TEXT 12 WORDS	100% MATCHING TEXT 12 WORDS
	for the purpose of recovery and concurrency control, we are only interested	
SA	DBMS Block 2.pdf (D149011992)	

189/319	SUBMITTED TEXT 40 WORDS	62% MATCHING TEXT 40 WORDS
	r 1 (A); w 1 (A); r 1 (B); w 1 (B); r 5 (B); w 5 (B); r 5 (C); w 5 (C);	
SA	MCA-Sem-II-Database Management System.pdf (D143652738)	

190/319	SUBMITTED TEXT 25 WORDS	70% MATCHING TEXT 25 WORDS
	Two operations in a schedule are said to conflict if they belong to different transactions, access the same data item, and at least one of	Two operations in a schedule are said to conflict if they belong to transactions, and they access the same item X, and if one of
W	https://present5.com/database-theory-jason-fan-outline-basic/	

191/319	SUBMITTED TEXT 29 WORDS	100% MATCHING TEXT 29 WORDS
	T 2 , T 3 , ..., T n is serializable if it is equivalent to some serial schedule of the same n transactions.	
SA	Database Management System Block 2.pdf (D164883642)	

192/319	SUBMITTED TEXT 40 WORDS	62% MATCHING TEXT 40 WORDS
	r 1 (A); w 1 (A); r 5 (B); w 5 (B); r 1 (B); w 1 (B); r 5 (C); w 5 (C);	
SA	MCA-Sem-II-Database Management System.pdf (D143652738)	

193/319	SUBMITTED TEXT 12 WORDS	95% MATCHING TEXT 12 WORDS
	effect on the database or on the other transactions in the	effect whatever on the database or on the other transactions in the
W	https://present5.com/database-theory-jason-fan-outline-basic/	

194/319	SUBMITTED TEXT 14 WORDS	96% MATCHING TEXT 14 WORDS
	conflict equivalent if the order of any two conflicting operations is same in both	conflict equivalent: If the order of any two conflicting operations is the same in both
W	https://present5.com/database-theory-jason-fan-outline-basic/	

195/319	SUBMITTED TEXT	27 WORDS	85%	MATCHING TEXT	27 WORDS

If a schedule S can be transformed into a schedule S′ by performing a series of swaps of non-conflicting operations, then S and S′ are conflict equivalent.

SA DBMS Block 2.pdf (D149011992)

196/319	SUBMITTED TEXT	24 WORDS	66%	MATCHING TEXT	24 WORDS

are conflict equivalent. The concept of conflict equivalence leads to the concept of conflict serializability. A schedule, which is conflict equivalent to some serial schedule,

SA BOOK SIZE.docx (D50092775)

197/319	SUBMITTED TEXT	18 WORDS	61%	MATCHING TEXT	18 WORDS

conflicting operations in both the schedules and the order of these operations in both the schedules is

SA Database System.pdf (D165747767)

198/319	SUBMITTED TEXT	34 WORDS	72%	MATCHING TEXT	34 WORDS
create an edge (T i → T j) in the precedence graph. 3. If T j executes a write operation after T i executes a read				create an edge (T i T j) in the precedence graph. 3. For each case in S where T j executes a write_item(X) after T i executes a read_	

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

199/319	SUBMITTED TEXT	34 WORDS	72%	MATCHING TEXT	34 WORDS
create an edge (T i → T j) in the precedence graph. 4. If T j executes a write operation after T i executes a write				create an edge (T i T j) in the precedence graph. 4. For each case in S where T j executes a write_item(X) after T i executes a write_	

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

200/319	SUBMITTED TEXT	15 WORDS	90%	MATCHING TEXT	15 WORDS
precedence graph (or serialization graph). A precedence graph is a directed graph G = (N,E)				Precedence graph (Serialization graph) A precedence graph is a directed graph G = (N,E)	

W https://slideplayer.com/slide/5143620/

201/319	SUBMITTED TEXT 17 WORDS	100% MATCHING TEXT 17 WORDS
	create an edge (T i → T j) in the precedence graph. 5.	create an edge (T i T j) in the precedence graph. 3.
W	http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf	

202/319	SUBMITTED TEXT 11 WORDS	87% MATCHING TEXT 11 WORDS
	precedence graph contains a cycle, the schedule S is not conflict serializable.	precedence graph contains a cycle, the schedule is not conflict serializable.
SA	Advanced DBMS.pdf (D166063498)	

203/319	SUBMITTED TEXT 12 WORDS	100% MATCHING TEXT 12 WORDS
	participating in the schedule S, create a node labelled T	
SA	Database System.pdf (D165747767)	

204/319	SUBMITTED TEXT 18 WORDS	100% MATCHING TEXT 18 WORDS
	an edge (T i → T j) exists in the precedence graph,	an edge T i →T j exists in the precedence graph
SA	Advanced DBMS.pdf (D166063498)	

205/319	SUBMITTED TEXT 32 WORDS	74% MATCHING TEXT 32 WORDS
	are said to be view equivalent if the schedules satisfy these conditions. 1. The same set of transactions participates in S and S′, and S and S′ include the same set of	are said to be view equivalent if the following three hold: The same set of transactions participate in S and S'; and S and S' include the same operations of
W	https://slideplayer.com/slide/5143620/	

206/319	SUBMITTED TEXT 14 WORDS	90% MATCHING TEXT 14 WORDS
	in schedule S, then T i must read the initial value of	in schedule S 1 , then transaction T i must also read the initial value of
SA	Advanced DBMS.pdf (D166063498)	

207/319	SUBMITTED TEXT	10 WORDS	100%	MATCHING TEXT	10 WORDS

in any serial schedule S′ that is equivalent to S.

SA BOOK SIZE.docx (D50092775)

208/319	SUBMITTED TEXT	28 WORDS	75%	MATCHING TEXT	28 WORDS

The concept of view equivalence leads to the concept of view serializability. A schedule S is said to be view serializable if it is view equivalent to some serial schedule.

SA BOOK SIZE.docx (D50092775)

209/319	SUBMITTED TEXT	12 WORDS	88%	MATCHING TEXT	12 WORDS

blind writes appear in a view serializable schedule that is not conflict serializable.

SA BOOK SIZE.docx (D50092775)

210/319	SUBMITTED TEXT	24 WORDS	85%	MATCHING TEXT	24 WORDS

T j has read the value of data item written by T i) must also be rolled back.

SA DBMS Block 2.pdf (D149011992)

211/319	SUBMITTED TEXT	39 WORDS	78%	MATCHING TEXT	39 WORDS

schedule S, a transaction T j reads a data item written by T i , then the commit operation of T i should appear before the commit operation of T j .

SA U234.pdf (D109498494)

212/319	SUBMITTED TEXT	15 WORDS	87%	MATCHING TEXT	15 WORDS

the transaction T 22 reads the value of data item Q written by

SA BOOK SIZE.docx (D50092775)

213/319	SUBMITTED TEXT	22 WORDS	76%	MATCHING TEXT	22 WORDS

a single transaction leads to a series of transaction rollbacks, is called cascading rollback. Cascading rollback requires undoing of significant amount of work

SA DBMS Block 2.pdf (D149011992)

214/319 **SUBMITTED TEXT** 42 WORDS **95%** **MATCHING TEXT** 42 WORDS

that T j reads a data item previously written by T i ; however, commit operation of T i appears before the read operation of T j . Every cascadeless schedule is also recoverable. 9.5

SA U234.pdf (D109498494)

215/319 **SUBMITTED TEXT** 13 WORDS **100%** **MATCHING TEXT** 13 WORDS

The access mode can be specified as READ ONLY or READ WRITE. The

SA MCA-Sem-II-Database Management System.pdf (D143652738)

216/319 **SUBMITTED TEXT** 22 WORDS **76%** **MATCHING TEXT** 22 WORDS

a transaction T i reads a set of rows from a table based on some condition specified in WHERE clause,

SA MCA-Sem-II-Database Management System.pdf (D143652738)

217/319 **SUBMITTED TEXT** 11 WORDS **100%** **MATCHING TEXT** 11 WORDS

allows UPDATE, INSERT, DELETE, and CREATE commands to be executed. 257

SA MCA-Sem-II-Database Management System.pdf (D143652738)

218/319 **SUBMITTED TEXT** 13 WORDS **100%** **MATCHING TEXT** 13 WORDS

A collection of operations that form a single logical unit of work	a collection of operations that form a single logical unit of work

SA Advanced DBMS.pdf (D166063498)

219/319 **SUBMITTED TEXT** 11 WORDS **100%** **MATCHING TEXT** 11 WORDS

Whenever a transaction is submitted to a DBMS for execution,	Whenever a transaction is submitted to a DBMS for execution,

W https://present5.com/database-theory-jason-fan-outline-basic/

220/319 **SUBMITTED TEXT** 16 WORDS **90%** **MATCHING TEXT** 16 WORDS

transactions and also preserve the order in which the instructions appear in each individual transaction. 13. A	transactions and must preserve the order in which the instructions appear in each individual transaction. A

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

221/319	SUBMITTED TEXT 26 WORDS	59% MATCHING TEXT 26 WORDS
	If an error occurs on any SQL statement, the entire transaction is rolled back. That is, any updated value would be restored to its original value.	

SA MCA-Sem-II-Database Management System.pdf (D143652738)

222/319	SUBMITTED TEXT 19 WORDS	86% MATCHING TEXT 19 WORDS
	A precedence graph is a directed graph consists of a set of nodes and a set of directed edges. 20.	a precedence graph is a directed 91 graph G = (N, E) that consists of a set of nodes N and a set of directed edges

SA Advanced DBMS.pdf (D166063498)

223/319	SUBMITTED TEXT 24 WORDS	79% MATCHING TEXT 24 WORDS
	transactions, access the same data item and at least one of them is write operation. 18. Two schedules are said to be conflict equivalent if	

SA BOOK SIZE.docx (D50092775)

224/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	precedence graph contains a cycle, the schedule is not conflict serializable. 21.	precedence graph contains a cycle, the schedule is not conflict serializable.

SA Advanced DBMS.pdf (D166063498)

225/319	SUBMITTED TEXT 19 WORDS	91% MATCHING TEXT 19 WORDS
	A schedule S is said to be view serializable if it is view equivalent to some serial schedule. 23.	A schedule S is said to be view serializable if it is view equivalent to a serial schedule.

W https://present5.com/database-theory-jason-fan-outline-basic/

226/319	SUBMITTED TEXT 13 WORDS	96% MATCHING TEXT 13 WORDS
	single transaction leads to a series of transaction rollbacks, is called cascading rollback.	single transaction leads to a series of transaction rollbacks is called Cascading rollback.

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

227/319	SUBMITTED TEXT 16 WORDS	76% MATCHING TEXT 16 WORDS
	If the precedence graph is acyclic then the schedule is considered to be conflict serializable.	

SA DBMS.docx (D142535242)

228/319	SUBMITTED TEXT 25 WORDS	71% MATCHING TEXT 25 WORDS
	If a transaction T i has acquired a shared lock on data item Q, T i can read but cannot write	If a transaction T i has obtained a shared-mode lock (denoted by S) 148 on item Q, then T i can read, but cannot write,
	SA Advanced DBMS.pdf (D166063498)	

229/319	SUBMITTED TEXT 41 WORDS	67% MATCHING TEXT 41 WORDS
	Suppose a transaction T i requests a lock of mode m 1 on a data item Q on which another transaction T j currently holds a lock of mode m 2 . If	Suppose that a transaction T i requests a lock of mode A on item Q on which transaction T j (T i ≠ T j) currently holds a lock of mode B. If
	SA Advanced DBMS.pdf (D166063498)	

230/319	SUBMITTED TEXT 41 WORDS	33% MATCHING TEXT 41 WORDS
	data item Q and another transaction, say T j , requests an exclusive lock on the data item Q. Clearly, T j has to wait until the transaction T i releases the shared lock on Q.	
	SA BOOK SIZE.docx (D50092775)	

231/319	SUBMITTED TEXT 15 WORDS	73% MATCHING TEXT 15 WORDS
	on the required data items. A lock is a variable associated with each data item	
	SA MCA-Sem-II-Database Management System.pdf (D143652738)	

232/319	SUBMITTED TEXT 16 WORDS	75% MATCHING TEXT 16 WORDS
	table, indexed on the data item identifier, to find the linked list for a data item.	table, indexed on the name of a data item, to find the linked list (if any) for a data item;
	SA Advanced DBMS.pdf (D166063498)	

233/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	waiting for the release of a data item held by	waiting for the release of a data – item held by
	W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288	

234/319	SUBMITTED TEXT 12 WORDS	71% MATCHING TEXT 12 WORDS
	just after the lock-X(Q) statement and still maintains the two-phase locking property. The	just after the lock- X (A) instruction, and still retain the two-phase locking property. The
	SA Advanced DBMS.pdf (D166063498)	

235/319	SUBMITTED TEXT 21 WORDS	52% MATCHING TEXT 21 WORDS
	this is a contradiction that T i is following two-phase locking. Thus, our assumption that S is non-serializable is wrong.	this is a contradiction of the assertion that T1 is using two phase locking protocol. Thus the assumption that graph is having a cycle is wrong

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

236/319	SUBMITTED TEXT 10 WORDS	95% MATCHING TEXT 10 WORDS
	a modification of two-phase locking called strict two-phase locking	a modification of two-phase locking called the strict two-phase locking

SA Advanced DBMS.pdf (D166063498)

237/319	SUBMITTED TEXT 15 WORDS	87% MATCHING TEXT 15 WORDS
	strict two-phase locking, a transaction does not release any of its exclusive-mode locks until	Strict Two-Phase locking : A transaction T does not release any of its exclusive (X) locks until

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

238/319	SUBMITTED TEXT 10 WORDS	95% MATCHING TEXT 10 WORDS
	two-phase locking, a transaction does not release any of its	Two-Phase locking : A transaction T does not release any of its

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

239/319	SUBMITTED TEXT 16 WORDS	80% MATCHING TEXT 16 WORDS
	This is because searching an index always starts at the root, and if any transaction	This is because searching an index always starts at the root, so if a transaction 161

SA Advanced DBMS.pdf (D166063498)

240/319	SUBMITTED TEXT 16 WORDS	92% MATCHING TEXT 16 WORDS
	a variant of the B + -tree called B-link tree. In B-link tree,	a variant of the B + -tree, called the B-link tree. In a B-link tree,

SA Advanced DBMS.pdf (D166063498)

241/319	SUBMITTED TEXT 25 WORDS	59% MATCHING TEXT 25 WORDS
	The point in the schedule at which the transaction successfully acquires its last lock is called the lock point of the transaction. For example, the transactions	

SA BOOK SIZE.docx (D50092775)

242/319	SUBMITTED TEXT 14 WORDS	66% MATCHING TEXT 14 WORDS
	Intention-shared (IS) lock is incompatible only with exclusive lock whereas intention-exclusive (IX) lock	intention shared (IS) lock conflicts only with an exclusive lock; an intention exclusive (IX) lock

SA Advanced DBMS.pdf (D166063498)

243/319	SUBMITTED TEXT 37 WORDS	64% MATCHING TEXT 37 WORDS
	with each data item Q. ? read_TS(Q): The read timestamp of Q; this is the largest timestamp among all the timestamps of transactions that have successfully executed read(Q). ? write_TS(Q): The write timestamp of Q; this is the largest	with each database item X two timestamp values: • 1. read_TS(X): the read timestamp of item X; this is the largest timestamp among all the timestamps of transactions that have successfully read item X. • 2. write_TS(X): the write timestamp of item X; this is the largest

W https://present5.com/database-theory-jason-fan-outline-basic/

244/319	SUBMITTED TEXT 10 WORDS	100% MATCHING TEXT 10 WORDS
	timestamp among all the timestamps of transactions that have successfully	timestamp among all the timestamps of transactions that have successfully

W https://present5.com/database-theory-jason-fan-outline-basic/

245/319	SUBMITTED TEXT 26 WORDS	57% MATCHING TEXT 26 WORDS
	written by T i , must be rolled back. Similarly, any other transaction T k , which may have used a value	written by T must also be rolled back. Similarly, any transaction T 2 that may have used a value

W https://present5.com/database-theory-jason-fan-outline-basic/

246/319	SUBMITTED TEXT 24 WORDS	61% MATCHING TEXT 24 WORDS
	data item written by T j , must also be rolled back, and so on. This situation is known as cascading rollback,	

SA 47F417BA15858476660.pdf (D123781188)

Ouriginal by Turnitin

247/319	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

the read operation is rejected and T i is rolled back.

SA U234.pdf (D109498494)

248/319	SUBMITTED TEXT	17 WORDS	82%	MATCHING TEXT	17 WORDS

and any other values it has acquired are expected to be inconsistent with the modified value of

SA DBMS_AIML_FINAL.pdf (D111167082)

249/319	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

the write operation is rejected and T i is rolled back.

SA U234.pdf (D109498494)

250/319	SUBMITTED TEXT	24 WORDS	46%	MATCHING TEXT	24 WORDS

Submitted	Matching
of the transaction. If the restarting of a transaction occurs several times, conflicting transactions need to be temporarily blocked which allows the transaction to finish.	of the long transaction. If a transaction is suffering from repeated conflicting transactions need to be temporarily blocked to enable the transaction to finish.

SA Advanced DBMS.pdf (D166063498)

251/319	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

the write operation is rejected and T i is rolled back.

SA U234.pdf (D109498494)

252/319	SUBMITTED TEXT	26 WORDS	100%	MATCHING TEXT	26 WORDS

Submitted	Matching
the value of Q that T i is producing was previously needed, and it had been assumed that the value would never be produced. 277 2.	the value of Q that T i is producing was previously needed, and it had been assumed that the value would never be produced.

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

253/319	SUBMITTED TEXT	14 WORDS	100% MATCHING TEXT	14 WORDS

the write operation is rejected and T i is rolled back.

SA U234.pdf (D109498494)

254/319	SUBMITTED TEXT	13 WORDS	100% MATCHING TEXT	13 WORDS

depending on whether it is a read-only or an update transaction. The

depending on whether it is a read-only or an update transaction. The

SA Advanced DBMS.pdf (D166063498)

255/319	SUBMITTED TEXT	12 WORDS	83% MATCHING TEXT	12 WORDS

by the timestamp ordering technique. Therefore, the value of timestamp Validation(T

by the timestamp-ordering technique, using the value of the timestamp Validation(T

SA Advanced DBMS.pdf (D166063498)

256/319	SUBMITTED TEXT	32 WORDS	34% MATCHING TEXT	32 WORDS

of the transaction be maintained by the system. The read_set of the transaction is the set of data items it reads and the write_set of the transaction is the set of data items

of the transaction to be rolled back is based on the following considerations: • The progress of the transaction and the number of data – items it has used and modified. • The amount of computing remaining for the transaction and the number of data – items

W http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf

257/319	SUBMITTED TEXT	18 WORDS	75% MATCHING TEXT	18 WORDS

the schedule shown in Figure 10.15 is view equivalent to the serial schedule of T 9

SA BOOK SIZE.docx (D50092775)

258/319	SUBMITTED TEXT	18 WORDS	89% MATCHING TEXT	18 WORDS

schedule must be equivalent to a serial schedule in which T i appears before T

schedule must be equivalent to a serial schedule in which transaction T j appears before T

SA Advanced DBMS.pdf (D166063498)

259/319	SUBMITTED TEXT	19 WORDS	76% MATCHING TEXT	19 WORDS

For every pair of transactions T i and T j such that TS(T

SA U234.pdf (D109498494)

260/319	SUBMITTED TEXT 14 WORDS	89% MATCHING TEXT 14 WORDS
	the value that it was supposed to read has already been overwritten. However, these	the value that it was supposed to read has already been overwritten. These

SA Advanced DBMS.pdf (D166063498)

261/319	SUBMITTED TEXT 56 WORDS	70% MATCHING TEXT 56 WORDS
	read_TS(Q i): The read timestamp of Q i ; this is the largest timestamp among all the timestamps of transactions that have successfully read Q i . ? write_TS(Q i): The write timestamp of Q i ; this is the timestamp of the transaction that	read_TS(X): the read timestamp of item X; this is the largest timestamp among all the timestamps of transactions that have successfully read item X. • 2. write_TS(X): the write timestamp of item X; this is the largest of all the timestamps of transactions that

W https://present5.com/database-theory-jason-fan-outline-basic/

262/319	SUBMITTED TEXT 26 WORDS	83% MATCHING TEXT 26 WORDS
	write_set of T j . Since T i completes its read phase before T j completes its read phase,	

SA DBMS Block 2.pdf (D149011992)

263/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	waiting for the release of a data item held by	waiting for the release of a data – item held by

W https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288

264/319	SUBMITTED TEXT 17 WORDS	100% MATCHING TEXT 17 WORDS
	T j is rolled back (T j is wounded by T	T j is rolled back (T j is wounded by T

SA Advanced DBMS.pdf (D166063498)

265/319	SUBMITTED TEXT 24 WORDS	69% MATCHING TEXT 24 WORDS
	the value of uncertified version of Q. However, once T i is ready to commit, it must acquire a certify lock on	

SA DBMS Block 2.pdf (D149011992)

266/319	SUBMITTED TEXT 13 WORDS	88% MATCHING TEXT 13 WORDS
	in the order in which the requests arrive. It uses a hash table	in the order in which the requests arrived. It uses a hash table,

SA Advanced DBMS.pdf (D166063498)

267/319	SUBMITTED TEXT 19 WORDS	88% MATCHING TEXT 19 WORDS
	strict two-phase, a transaction does not release any of its exclusive-mode locks until that transaction commits or aborts. 9. In	Strict Two-Phase locking : A transaction T does not release any of its exclusive (X) locks until that transaction commits or aborts. In
W	http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf	

268/319	SUBMITTED TEXT 11 WORDS	95% MATCHING TEXT 11 WORDS
	two-phase locking, a transaction does not release any of its	Two-Phase locking : A transaction T does not release any of its
W	https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288	

269/319	SUBMITTED TEXT 14 WORDS	95% MATCHING TEXT 14 WORDS
	a variant of the B + -tree called B-link tree, in	a variant of the B + -tree, called the B-link tree. In
SA	Advanced DBMS.pdf (D166063498)	

270/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	waiting for the release of a data item held by	waiting for the release of a data – item held by
W	https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288	

271/319	SUBMITTED TEXT 12 WORDS	100% MATCHING TEXT 12 WORDS
	helps in reducing the time taken to recover from a crash ?	
SA	MCA-Sem-II-Database Management System.pdf (D143652738)	

272/319	SUBMITTED TEXT 16 WORDS	73% MATCHING TEXT 16 WORDS
	on the required data items. 3. A lock is a variable associated with each data item	
SA	MCA-Sem-II-Database Management System.pdf (D143652738)	

273/319	SUBMITTED TEXT 18 WORDS	75% MATCHING TEXT 18 WORDS
	to determine whether the disk page containing the data item resides in the cache. If it is not	
SA	47F417BA15858476660.pdf (D123781188)	

274/319	SUBMITTED TEXT 36 WORDS	70% MATCHING TEXT 36 WORDS
	the cache. Sometimes it may be necessary to replace some of the disk pages to create space for the new pages. Any page-replacement strategy such as least recently used (LRU) or first-in-first- out (FIFO) can be used	

SA 47F417BA15858476660.pdf (D123781188)

275/319	SUBMITTED TEXT 15 WORDS	100% MATCHING TEXT 15 WORDS
	Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a	

SA 47F417BA15858476660.pdf (D123781188)

276/319	SUBMITTED TEXT 26 WORDS	62% MATCHING TEXT 26 WORDS
	have been force-written to the disk. 2. The transaction is not allowed to commit until all its REDO-type and UNDO-type log records have been force- written to	have been force-written to the disk. The commit operation of a transaction can not be completed until all the REDO-type and UNDO-type log records for that transaction have been force written to

W https://slideplayer.com/slide/5143620/

277/319	SUBMITTED TEXT	185 WORDS	99%	MATCHING TEXT	185 WORDS

RECOVERY RELATED STEPS DURING NORMAL EXECUTION When a DBMS is restarted after a crash, the control is given to the recovery manager, which is responsible to bring the database to a consistent state. To enable it to perform its task in the event of failure, the recovery manager maintains some information during the normal execution of transactions. It maintains a system log of all the modifications to the 288 database and stores it on the stable storage, which is guaranteed to survive failures. The stable storage is implemented by storing the database on a number of separate non- volatile storage devices such as disks or tapes (perhaps in different locations). Information residing in stable storage is assumed to be never lost (in real life never cannot be guaranteed). The amount of work involved during recovery depends on the changes made by the committed transactions that have not been written to the disk at the time of crash. To reduce the time to recover from a crash, the DBMS periodically force- write all the modified buffer pages during normal execution. A process called checkpointing also helps in reducing the time taken to recover from a crash. 11.3.1

SA MCA-Sem-II-Database Management System.pdf (D143652738)

278/319	SUBMITTED TEXT	24 WORDS	42%	MATCHING TEXT	24 WORDS

The UNDO-type log entry includes the before-image (BFIM) of the data item which is used to undo the effect of an operation on the

SA 47F417BA15858476660.pdf (D123781188)

279/319	SUBMITTED TEXT	20 WORDS	65%	MATCHING TEXT	20 WORDS

a transaction T 1 that transfers $100 from account A to account B. Suppose accounts A and B

SA DBMS_AIML_FINAL.pdf (D111167082)

280/319	SUBMITTED TEXT	39 WORDS	100%	MATCHING TEXT	39 WORDS

read(A); A:= A - 100; 289 write(A); read(B); B:= B + 100; write(B); The

SA MCA-Sem-II-Database Management System.pdf (D143652738)

281/319	SUBMITTED TEXT 11 WORDS	95% MATCHING TEXT 11 WORDS
	read operations are recorded in the log only to determine whether	
SA	47F417BA15858476660.pdf (D123781188)	

282/319	SUBMITTED TEXT 13 WORDS	88% MATCHING TEXT 13 WORDS
	The checkpoint approach is an additional component of the logging scheme. A checkpoint	The checkpoint scheme is an additional component of the logging scheme. A checkpoint
W	http://www.vmsclass.in/wp-content/uploads/2018/09/MCQ.pdf	

283/319	SUBMITTED TEXT 14 WORDS	92% MATCHING TEXT 14 WORDS
	Hence, there is no need to record any read operations in the log.	
SA	47F417BA15858476660.pdf (D123781188)	

284/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	the database to the consistent state that existed before the	the database to the consistent state that existed before the
SA	Advanced DBMS.pdf (D166063498)	

285/319	SUBMITTED TEXT 15 WORDS	100% MATCHING TEXT 15 WORDS
	and this type of failure can occur any number of times before the recovery is	and this type of failure can occur any number of times before the recovery is
W	https://dokumen.site/download/question-solution-ac14-sol-a5b39efca29288	

286/319	SUBMITTED TEXT 17 WORDS	58% MATCHING TEXT 17 WORDS
	redone in case of a failure as all updates are already recorded in the database on disk.	
SA	47F417BA15858476660.pdf (D123781188)	

287/319	SUBMITTED TEXT 20 WORDS	67% MATCHING TEXT 20 WORDS
	transaction fails before reaching its commit point no undo is required as the transaction has not affected the database on	
SA	47F417BA15858476660.pdf (D123781188)	

288/319	SUBMITTED TEXT	14 WORDS	89% MATCHING TEXT	14 WORDS
	is required in case the system fails after the transaction commits but before all		is needed in case the system fails after the transaction commits but before all	

W https://present5.com/database-theory-jason-fan-outline-basic/

289/319	SUBMITTED TEXT	15 WORDS	96% MATCHING TEXT	15 WORDS
	transaction must be restarted either automatically by the recovery process or manually by the user.			

SA 47F417BA15858476660.pdf (D123781188)

290/319	SUBMITTED TEXT	17 WORDS	100% MATCHING TEXT	17 WORDS
	are executed serially in the order T 1 followed by T 2 .			

SA MCA-Sem-II-Database Management System.pdf (D143652738)

291/319	SUBMITTED TEXT	51 WORDS	89% MATCHING TEXT	51 WORDS
	T 1 , start] [T 1 , A, 2000] [T 1 , B, 1500] [T 1 , commit] [T 2 , start] [T 2 , C, 500] [T 2 , commit]		T0 start&lt; &gt;T0, A, 1000, 950&lt; To, B, 2000, 2050 &gt;T0 commit&lt; &gt;T1 start&lt; &gt;T1, C, 700, 600&lt; &gt;T1 commit&lt;	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

292/319	SUBMITTED TEXT	51 WORDS	89% MATCHING TEXT	51 WORDS
	T 1 , start] [T 1 , A, 2000, 1900] [T 1 , B, 1500, 1600] [T 1 , commit] [T 2 , start] [T 2 ,C,500,700] [T 2 , commit]		T0 start&lt; &gt;T0, A, 1000, 950&lt; To, B, 2000, 2050 &gt;T0 commit&lt; &gt;T1 start&lt; &gt;T1, C, 700, 600&lt; &gt;T1 commit&lt;	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

293/319	SUBMITTED TEXT	14 WORDS	75% MATCHING TEXT	14 WORDS
	the database. This technique assumes that only one transaction is active at a time		The shadow-database scheme: o Assumes that only one transaction is active at a time.	

W https://vemu.org/uploads/lecture_notes/15_02_2021_1663694760.pdf

294/319	SUBMITTED TEXT 12 WORDS	87% MATCHING TEXT 12 WORDS
	Shadow paging considers the database to be made up of fixed-size	Shadow paging considers the database to be made up of n number of fixed-size
	SA Advanced DBMS.pdf (D166063498)	

295/319	SUBMITTED TEXT 28 WORDS	50% MATCHING TEXT 28 WORDS
	with n entries is constructed in which the i th entry in the page table points to the i th database page on the	with n entries is constructed, where the i th entry points to the i th database page on disk. The
	SA Advanced DBMS.pdf (D166063498)	

296/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	are recorded in the database on the disk before the transaction	
	SA Sem III_ BCA_B21CA05DC.docx (D165628090)	

297/319	SUBMITTED TEXT 16 WORDS	86% MATCHING TEXT 16 WORDS
	When a transaction starts, the current page table is copied into a shadow page table (	When a transaction begins executing, the current page table is copied into a page table,
	W https://present5.com/database-theory-jason-fan-outline-basic/	

298/319	SUBMITTED TEXT 16 WORDS	70% MATCHING TEXT 16 WORDS
	shadow page table is then saved on the disk and the current page table is	shadow page table, which is then saved on the disk. During transaction execution, the shadow page table is
	W https://present5.com/database-theory-jason-fan-outline-basic/	

299/319	SUBMITTED TEXT 11 WORDS	100% MATCHING TEXT 11 WORDS
	pages updated by the transactions, two versions are kept. The	pages updated by the transaction, two versions are kept. The
	SA Advanced DBMS.pdf (D166063498)	

300/319	SUBMITTED TEXT 12 WORDS	95% MATCHING TEXT 12 WORDS
	to commit before all its changes are written to the database. In	
	SA DBMS Block 2.pdf (D149011992)	

301/319	SUBMITTED TEXT	25 WORDS	62%	MATCHING TEXT	25 WORDS

the write operations of all the transactions in the commit list are redone in the order in which they were written to the log. The

the WRITE operations of the committed transactions from the in the order in which they were written to the log. The

W https://present5.com/database-theory-jason-fan-outline-basic/

302/319	SUBMITTED TEXT	23 WORDS	86%	MATCHING TEXT	23 WORDS

In general, the greater the degree of concurrency, the more time-consuming the task of recovery becomes. Consider a system in which strict two-phase locking

SA 47F417BA15858476660.pdf (D123781188)

303/319	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

undone in the reverse of the order in which they were written

SA 47F417BA15858476660.pdf (D123781188)

304/319	SUBMITTED TEXT	21 WORDS	71%	MATCHING TEXT	21 WORDS

If the item is found in the list, it is not redone again as its last value has already been recovered. 11.6

SA 47F417BA15858476660.pdf (D123781188)

305/319	SUBMITTED TEXT	23 WORDS	56%	MATCHING TEXT	23 WORDS

actions of database system prior to the crash to bring the database to the state which existed at the time of the crash.

SA MCA-Sem-II-Database Management System.pdf (D143652738)

306/319	SUBMITTED TEXT	17 WORDS	66%	MATCHING TEXT	17 WORDS

file. The new file is assigned a number higher by 1 than the previous log file.

file; the new log file has a file number that is higher by 1 than the previous log file.

SA Advanced DBMS.pdf (D166063498)

307/319	SUBMITTED TEXT	14 WORDS	96% MATCHING TEXT	14 WORDS
	the LSN consists of a file number and an offset within the file.		The LSN then consists of a file number and an offset within the file.	

SA Advanced DBMS.pdf (D166063498)

308/319	SUBMITTED TEXT	23 WORDS	50% MATCHING TEXT	23 WORDS
	log records with LSN less than or equal to pageLSN of the page should not be executed as their actions are already reflected		log records with LSN less than or equal to the PageLSN of a page should not be executed on the page, since their actions are already reflected	

SA Advanced DBMS.pdf (D166063498)

309/319	SUBMITTED TEXT	13 WORDS	100% MATCHING TEXT	13 WORDS
	that needs to be undone next, when the transaction is being rolled back.		that needs to be undone next, when the transaction is being rolled back.	

SA Advanced DBMS.pdf (D166063498)

310/319	SUBMITTED TEXT	16 WORDS	66% MATCHING TEXT	16 WORDS
	transactions that were not committed at the time of the crash. ? Logging changes during undo:			

SA MCA-Sem-II-Database Management System.pdf (D143652738)

311/319	SUBMITTED TEXT	16 WORDS	85% MATCHING TEXT	16 WORDS
	the database and the log are copied periodically onto a cheap storage medium such as magnetic tapes.			

SA 47F417BA15858476660.pdf (D123781188)

312/319	SUBMITTED TEXT	11 WORDS	95% MATCHING TEXT	11 WORDS
	site. The remote site must be physically separated from the primary		site. The remote backup site must be physically separated from the primary—	

SA Advanced DBMS.pdf (D166063498)

Ouriginal
by Turnitin

313/319	SUBMITTED TEXT	13 WORDS	80% MATCHING TEXT	13 WORDS
	log records from the primary site to the remote site where the backup		log records from the primary site to the remote backup site. The remote backup	

SA Advanced DBMS.pdf (D166063498)

314/319	SUBMITTED TEXT	11 WORDS	90% MATCHING TEXT	11 WORDS
	all the transactions that have been executed since the last backup.			

SA 47F417BA15858476660.pdf (D123781188)

315/319	SUBMITTED TEXT	14 WORDS	100% MATCHING TEXT	14 WORDS
	Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a			

SA 47F417BA15858476660.pdf (D123781188)

316/319	SUBMITTED TEXT	11 WORDS	100% MATCHING TEXT	11 WORDS
	the database to the consistent state that existed before the		the database to the consistent state that existed before the	

SA Advanced DBMS.pdf (D166063498)

317/319	SUBMITTED TEXT	17 WORDS	85% MATCHING TEXT	17 WORDS
	maintains a system log of all modifications to the database and stores it on the stable storage. 9.			

SA MCA-Sem-II-Database Management System.pdf (D143652738)

318/319	SUBMITTED TEXT	21 WORDS	100% MATCHING TEXT	21 WORDS
	To reduce the time to recover from a crash, the DBMS periodically force-write all the modified buffer pages during normal execution			

SA MCA-Sem-II-Database Management System.pdf (D143652738)

319/319	SUBMITTED TEXT	13 WORDS	76% MATCHING TEXT	13 WORDS
	is based on three principles including write-ahead logging, repeating history during redo,			

SA MCA-Sem-II-Database Management System.pdf (D143652738)