Document Information

Analyzed document	Advanced DBMS.pdf (D166063498)
Submitted	2023-05-06 06:48:00
Submitted by	Mumtaz B
Submitter email	mumtaz@code.dbuniversity.ac.in
Similarity	10%
Analysis address	mumtaz.dbuni@analysis.urkund.com

Sources included in the report

W	URL: https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019M... Fetched: 2021-05-01 09:42:28	58
SA	**47F417BA15858476660.pdf** Document 47F417BA15858476660.pdf (D123781188)	23
W	URL: http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc Fetched: 2022-11-22 00:17:32	82
SA	**DBMS_AIML_FINAL.pdf** Document DBMS_AIML_FINAL.pdf (D111167082)	39

Entire Document

Advanced Database Management System
Advanced Database Management System
Table of Contents Introduction Module 1 - Introduction & Types Of Databases
Unit: 01 Introduction 1.0 Introduction 1.1 Unit Objectives 1.2 Significance of Databases 1.2.1 Purpose of Database Systems 1.2.2 Components of DBMS Environment 1.3 Database System Applications 1.4 Advantages of DBMS 1.5 Disadvantages of DBMS 1.6 Summary 1.7 Key Terms 1.8 Check Your Progress
Unit: 02 Different types of databases 2.0 Introduction 2.1 Unit Objectives 2.2
Data Models 2.2.1 Object-based Data Models
2.2.2 Record-based Data Models 2.2.3 Physical Data Models 2.2.4 Conceptual Modeling 2.3
Relational
Database 2.4
Distributed Databases 2.5 Centralized Databases 2.6 Difference between Centralized and Distributed Databases 2.7 Summary 2.8 Key Terms 2.9 Check Your Progress
Module II Normalization & Its Forms
Unit: 03 Normalization 3.0 Introduction 3.1 Unit Objectives 3.2 Purpose of Normalization 3.3 Functional Dependency 3.4 Anomalies in a Database 3.5
The
Normalization Process 3.5.1
First normal form (1NF) 3.5.2 Second normal form (2NF) 3.5.3 Third normal form (3
NF) 3.6

Summary 3.7 Key Terms
3.8 Check Your Progress
Unit: 04 Other Normal Forms 4.0 Introduction 4.1 Unit Objectives 4.2 The Boyce-Codd
normal form (BCNF) 4.3 Multi-Valued Dependency (MVD) 4.4 Fourth Normal form (4NF) 4.5 Fifth normal form (5NF) 4.6 Database Design 4.7 Denormalization 4.8 Summary 4.9 Key Terms 4.10 Check Your Progress
Module III - Transaction Processing & Database Recovery Management
Unit: 05 Transaction Processing 5.0 Introduction 5.1 Unit Objectives 5.2 Basics of Transaction Processing 5.3 Serializability and recoverability 5.4 Deadlock Handling 5.4.1 Deadlock Prevention 5.4.2 Deadlock Detection & Recovery
5.5 Multilevel Transaction 5.6 Real- Time Transaction Systems 5.7 Long- Duration Transactions 5.8 Summary 5.9 Key Terms 5.10 Check Your Progress
Unit: 06 Database Recovery Management 6.0 Introduction 6.1 Unit Objectives 6.2 Need for Recovery 6.3 Recovery & Transactions 6.4 Provisions for Recovery 6.5 Failure Classification 6.6 Recovery Techniques 6.7 Remote Backup Systems 6.8 Summary 6.9 Key Terms 6.10 Check Your Progress
Module IV - Concurrency Control & Locking Systems
Unit: 07 Concurrency Control 7.0 Introduction 7.1 Unit Objectives
7.2 Need for Concurrency Control 7.3 Concurrency Control by Timestamps 7.4 Concurrency Control by Validation 7.5 Multiversion Schemes 7.6 Snapshot Isolation 7.7 Summary 7.8 Key Terms 7.9 Check Your Progress
Unit: 08 Locking Systems 8.0 Introduction 8.1 Unit Objectives 8.2 Locking Protocols 8.2.1 Two-Phase Locking (2PL) 8.3 Implementation of Locking 8.4 Granularity of Data Items 8.4.1 Multiple Granularity Locking 8.5 Concurrency control in index structures using locks 8.6 Other Concurrency Control Issues 8.6.1 Insertion, Deletion, and Phantom Records 8.6.2 Interactive Transactions 8.6.3 Latches 8.7 Summary 8.8 Key Terms
8.9 Check Your Progress
Module V - Object-Oriented DBMS
Unit: 09 Object-oriented DBMS: Concept & Design 9.0 Introduction 9.1 Unit Objectives 9.2 Overview of Object-oriented paradigm 9.3 Object Identity & Object Structure 9.4 Type hierarchies and inheritance 9.5 Object-Oriented Data Models 9.6 Persistent programming languages 9.7 Summary 9.8 Key Terms 9.9 Check Your Progress
Unit: 10 Other Concepts of Object-Oriented Databases 10.0 Introduction 10.1 Unit Objectives 10.2 Object-oriented database design 10.3 Persistence in OODBMSs 10.4 Encapsulation 10.5 Complex Objects 10.6 Issues in OODBMS 10.7 Advantages and Disadvantages of OODBMSs
10.8 Summary 10.9 Key Terms 10.10 Check Your Progress
Module VI - Distributed Database
Unit: 11 Distributed Database 11.0 Introduction 11.1 Unit Objectives 11.2 Basics of Distributed Database 11.3 DDBMS architectures 11.4 Homogeneous and Heterogeneous databases 11.5 Distributed data storage 11.6 Advantages & Disadvantages of Data Distribution 11.7 Summary 11.8 Key Terms 11.9 Check Your Progress
Unit: 12 Other concepts of Distributed Databases 12.0 Introduction 12.1 Unit Objectives 12.2 Distributed transactions 12.3 Concurrency control & recovery in distributed databases 12.3.1 Distributed Concurrency Control 12.3.2 Distributed Recovery
12.4 Directory systems 12.5 Commit Protocols & Availability 12.6 Data Allocation and Fragmentation 12.7 Distributed database transparency 12.8 Summary 12.9 Key Terms 12.10 Check Your Progress
Module VII Object Relational & Extended Relational Databases
Unit: 13 Introduction to Relational Databases 13.0 Introduction 13.1 Unit Objectives 13.2 Basic Concept of Relational Databases 13.3 Relational Database Design 13.4 Integrity Constraints 13.5
Standards for OODBMS 13.6 Summary 13.7 Key Terms 13.8 Check Your Progress
Unit: 14 Products and applications 14.0 Introduction 14.1 Unit Objectives 14.2 Overview of object model of ODMG 14.3 Object Definition & Query Language 14.4 An overview of SQL3 14.5 Nested relations and collections 14.6 Implementation issues for extended type 14.7 Comparing OODBMS & ORDBMS 14.8 Summary 14.9 Key Terms 14.10 Check Your Progress
Module VIII Advanced Databases
Unit: 15 Introduction to Advanced Databases 15.0 Introduction 15.1 Unit Objectives 15.2 Active database 15.3 Applications of active database 15.4 Temporal database 15.5 Spatial database 15.6 Summary 15.7 Key Terms 15.8 Check Your Progress
Unit: 16 Multimedia databases 16.0 Introduction 16.1 Unit Objectives
16.2 Concept of multimedia databases 16.3 Automatic Analysis of Images 16.4 Object Recognition in Images 16.5 Semantic Tagging of Images 16.6 Analysis of Audio Data Sources 16.7 Introduction to Mobile Databases 16.8 Summary 16.9 Key Terms 16.10 Check Your Progress
Appendix- I ARIES Appendix- II Concurrency Control & Recovery in Oracle
Introduction

64% MATCHING BLOCK 1/202 W

Databases and database systems have become an essential component of a modern lifestyle. Most of us encounter several activities every day that involve some interaction with a database.

Whether it is a

83% MATCHING BLOCK 5/202 W

bank deposit or withdraw funds, a hotel or airline reservation, a computerized library catalog to search for a bibliographic item, or

online purchasing; our everyday

100% MATCHING BLOCK 2/202 W

activities will involve someone or some computer program accessing a database.

88% MATCHING BLOCK 3/202 W

In the past few years, advances in technology have led to exciting new applications of database systems.

The proliferation of social media sites require the creation of huge databases that store non-traditional data. New types of database systems, often referred to as big data storage systems, or NOSQL systems, have been created to manage data for social media applications. To simplify learning, the study material is divided into eight modules each containing two units for the relevant topics. Module-1 provides the basic introduction to database management systems and its types. It explains the significance and purpose of different database systems. The unit also discusses different types of data models used for different types of databases. Module-2 describes the normalization feature of Database management systems (DBMS). The normal forms including 1NF, 2NF, 3NF, 4NF, BCNF, and 5NF are explained in detail. Module-3 elaborates the transaction processing techniques and its applications in the DBMS. This module also includes the unit explaining the database recovery management for retrieving the data if it is lost due to any reason.
Module-4 depicts the concept of concurrency control in a DBMS. It also explains different methods of concurrency control including locking systems, timestamps, validation, etc. The emerging trends in DBMS have led us to advanced DBMSs. Module-5 focuses on the object-oriented based databases, its basic concept and features. Module-6 is based on the study of Distributed databases, its basic concept, features, and applications. Module-7 focuses on the object relational and extended relational databases. Different applications of ORDBMS are also depicted in this module. ORDBMS is a combination of the features of relational DBMS and Object DBMS. Module-8 illustrates the advanced databases like active databases, temporal databases, spatial databases, multimedia databases, and mobile databases. Distinct features and applications of these advanced databases are also discussed in detail. This content is designed comprehensively and follows a simple approach, keeping in mind the syllabus of the program. It exhilarates interest and is sure to stimulate knowledge among the readers. Numerous figures and tables, key terms help in simplifying learning about the subject. The 'Check Your Progress' section intends the readers to test their knowledge. It is hoped that the language and the content demonstration is coherent to the readers and will enhance their learning in the best way possible. Keep Learning!
1 MODULE- I INTRODUCTION & TYPES OF DATABASES
2
3

Unit: 01 Introduction Structure 1.0 Introduction 1.1 Unit Objectives 1.2 Significance of Database 1.2.1 Purpose of Database Systems 1.2.2 Components of DBMS Environment 1.3 Database System Applications 1.4 Advantages of different Database Management systems 1.5 Disadvantages of different Database Management systems 1.6 Summary 1.7 Key Terms 1.8 Check Your Progress 1.0 Introduction A database management system (DBMS) is referred to as a collection of related data. It usually also includes a set of programs to access the relevant data. The primary aim of DBMS is to store and retrieve the relevant information in an efficient and convenient manner. The database systems are designed to manage a large amount of information. Defining structures for data storage and mechanisms for data retrieval, both are included in database management. Additionally, DBMS also ensures the security of information stored. For example, a telephone directory is the simplest example of a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications. Defining a database involves specifying
the data types, structures, and
4 constraints for the data to be stored in the database.
Constructing the database
is the process of storing the data itself on some storage medium that is controlled by the DBMS. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the real world, and generating reports from the data. This unit focuses on the significance of databases, various merits and demerits of DBMS, and basic database system applications. 1.1 Unit Objectives After completing this unit, the reader will be able to: ● Discuss the significance of the database. ● Understand the fundamentals of database system applications. ● Illustrate the advantages and disadvantages of DBMS. 1.2 Significance of Database
Indeed
the importance of the database system has increased in recent years with significant developments in hardware capability, hardware capacity, and communications, including the emergence of the Internet, electronic commerce, business intelligence, mobile communications, and grid computing.
We are already aware that
a
database is
a collection of relevant data. Database management system (DBMS) is the software that manages and controls access to the
database. A database application is a program that interacts with the database
during its execution. The database system is a collection of application programs that interact with the database along with
the DBMS and the database itself.
DBMS provides the following facilities: • DBMS allows users to define the database, usually through a
Data
5 Definition Language (DDL)
by specifying
the data types, structures, and constraints on the data to be stored in the database. •
DBMS allows users to
insert, update, delete, and retrieve data from the database,
usually through Data Manipulation Language (DML).
All the
data and data descriptions have a central repository that
allows
the DML to provide a general inquiry facility to this data, called a query language.
In a query language,
the user has to work with a fixed set of queries, causing major software management problems. The most common query language is the Structured Query Language (SQL). •
DBMS provides controlled access to the database. •
When we analyze the information needs of an organization, we attempt to identify entities, attributes, and relationships.
An
entity is a distinct object (a person, place, thing, concept, or event)
in the organization that is to be represented in the database. An
attribute is a property that describes some aspect of the object that we wish to record,
and a relationship is an association between entities. 1.2.1

Purpose of Database Systems Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps
the

83%	MATCHING BLOCK 7/202	SA	47F417BA15858476660.pdf (D123781188)

information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including

93%	MATCHING BLOCK 4/202	W

One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files,

programs to: • Add new students, instructors, and courses. • Register students for courses and generate class rosters. • Assign grades to students, compute grade point averages (GPA) , and
6 generate transcripts.

100%	MATCHING BLOCK 6/202	W

New application programs are added to the system as the need arises.

100%	MATCHING BLOCK 8/202	SA	47F417BA15858476660.pdf (D123781188)

New application programs are added to the system as the need arises. For example, suppose that

a university decides to create a new major (say, computer science). As a result, the university creates a new department and creates new permanent files (or adds information to existing files) to record information about all the instructors in the department, students in that major, course offerings, degree requirements, etc. The university

100%	MATCHING BLOCK 11/202	SA	47F417BA15858476660.pdf (D123781188)

may have to write new application programs to deal with

rules specific to the new major. New application programs may also have to be written to handle new rules in the university.

98%	MATCHING BLOCK 17/202	SA	47F417BA15858476660.pdf (D123781188)

Thus, as time goes by, the system acquires more files and more application programs. This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in a file-processing system has a number of major disadvantages: • Data redundancy and inconsistency Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system. Difficulty in accessing data. 7 •

68% **MATCHING BLOCK 9/202** W

in a file-processing system has a number of major disadvantages: • Data redundancy and inconsistency Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (

100% **MATCHING BLOCK 10/202** W

higher storage and access cost. In addition, it may lead to data inconsistency;

99% **MATCHING BLOCK 23/202** SA 47F417BA15858476660.pdf (D123781188)

Difficulty in accessing data Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use. • Data isolation Because data is scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult. • Integrity problems The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is 8 compounded when constraints involve several data items from different files. • Atomicity problems A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system. • Concurrent-access anomalies For the sake of the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment,

82% **MATCHING BLOCK 12/202** W

file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

75% **MATCHING BLOCK 13/202** W

data is scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data

91% **MATCHING BLOCK 14/202** W

Integrity problems The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below

100% MATCHING BLOCK 15/202 W

Developers enforce these constraints in the system by adding appropriate code in the

97% MATCHING BLOCK 16/202 W

application programs. However, when new constraints are added, it is difficult to change the programs to enforce them.

100% MATCHING BLOCK 20/202 W

A computer system, like any other mechanical or electrical device, is subject to failure.

91% MATCHING BLOCK 18/202 W

device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic —it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system. • Concurrent-access anomalies For the sake of

100% MATCHING BLOCK 19/202 W

overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.

the

99% MATCHING BLOCK 26/202 SA 47F417BA15858476660.pdf (D123781188)

interaction of concurrent updates may result in inconsistent data. Consider bank account A, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult 9 to provide because data may be accessed by many different application programs that have not been coordinated previously. • Security problems Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a bank enterprise as a running example of a typical data-processing application found in a corporation. 1.2.2 Components of

100% MATCHING BLOCK 21/202 W

Security problems Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about

77% MATCHING BLOCK 22/202 W

various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an

DBMS Environment Five
major components in the DBMS environment can be identified:
hardware, software, data, procedures, and people,
as illustrated in Figure 1.1.
Figure 1.1
Components of DBMS environment (Source-
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition,
Chapter- 1, Page No.- 66)
Hardware The
DBMS and the applications require hardware to run. The hardware can range from a single personal computer to a single mainframe or a network of computers. The particular hardware depends on the
organization's requirements
10 and the DBMS used. Some DBMSs run only on particular hardware or operating systems, while others run on a wide
variety of hardware and operating systems. A DBMS requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance.
Software The software component comprises the DBMS software itself and the application programs, together with the operating system,
including network software if the DBMS is being used over a network. Typically, application programs are written in a third-generation programming language (3GL), such as C, C++, C#, Java, Visual Basic, COBOL, Fortran, Ada, or Pascal, or a fourth-generation language (4GL), such as SQL, embedded in a third-generation language.
The target DBMS may have its own fourth-generation tools that allow rapid development of applications through the provision of
non-procedural
query languages, reports generators, forms generators, graphics generators, and application generators. The use of fourth-generation tools can improve productivity significantly and produce programs that are easier to maintain. Data
The most important component of the DBMS environment is Data. In figure 1.1, we can observe that the data acts as a bridge between the machine components and human components.
Procedures Procedures are
the
instructions and rules that govern the design and use of the database. The
documented procedures on how to use or run the system
are required for the users of
the
system. These may consist of instructions on how to: Log on to the DBMS, Use a particular DBMS facility or application program, Start and stop the DBMS, Make backup copies of the database, Handle hardware or software failures. This may include procedures on how to identify the failed component, how to fix the failed component, and following the repair of the fault, how to recover the database, Change the structure of a table, reorganize the
11 database across multiple disks, improve performance, or archive data to secondary storage.
People The final component is the people or the users involved with
the system. We
can identify
four

distinct types of people who participate
in the DBMS environment: data and database administrators, database designers, application developers,
and end-users. •
Data
and Database Administrators: •
The
database and the DBMS are corporate resources that must be managed like any other resource. Data
and database administration are the roles generally associated with the management and control of a DBMS and its data.
The
Data Administrator (DA) is responsible for the management of the data resource, including database planning;
development and
maintenance of standards, policies, and procedures; and conceptual/logical
database
design. The
DA consults with and advises senior managers, ensuring that the direction of database development will ultimately
support corporate objectives.
The Database Administrator (DBA) is responsible for the
physical realization of the database, including physical database design and implementation, security and integrity
control,
maintenance of the operational system, and ensuring satisfactory performance of the applications for users.
The role of the DBA is more technically oriented than the role of the DA, requiring detailed knowledge of the target
DBMS and the system environment. In some organizations there is no distinction between these two roles; in others, the
importance of the corporate resources is reflected in the allocation of teams of staff dedicated to each of these roles.
Database
Designers: In large database design projects, we can distinguish between two types of designers:
logical
database designers and physical
database designers. The
logical database designer is concerned with identifying the data (that is, the entities and attributes),
the relationships
12 between
the
data, and the
constraints on the data that is to be stored in the database.
The logical database
designer must have a thorough and
complete understanding of the organization's data and any constraints on this data (
the constraints are sometimes called business rules). These constraints
describe the main characteristics of the data as viewed by the organization.
The
physical database designer decides how the logical database design is to be physically realized.
This involves
mapping the logical database design into a set of tables and integrity constraints; selecting specific storage structures
and access methods for the data to achieve
a
good performance; designing any security measures required on the data.
The physical database designer must be capable of selecting a suitable storage strategy that takes account of usage.
Whereas conceptual and logical database design is concerned with the what, physical database design is concerned with
the how. •
Application Developers: •
Once the database has been implemented, the application programs that provide the required functionality for the
end- users must be implemented. This is the responsibility of the application developers.
Typically, the application developers
work from a specification produced by systems analysts. Each program contains statements that request the DBMS to
perform some operation on the database, which includes retrieving data, inserting, updating, and deleting data. The
programs may be written in a third-generation or fourth-generation programming language.

End-Users: The end-users are the "clients" of the database, which has been designed and implemented and is being maintained to serve their information needs. 1.3
Database System Applications Databases form an essential part of every enterprise today, storing not only types
13 of information that are common to most enterprises but also information that is specific to the category of the enterprise. The Internet revolution of the late 1990s sharply increased direct user access to databases. Organizations converted many of their phone interfaces to databases into Web interfaces and made a variety of services and information available online. For instance, when you access an online bookstore and browse a book or music collection, you are accessing data stored in a database. When you enter an order online, your order is stored in a database. When you access a bank website and retrieve your bank balance and transaction information, the information is retrieved from the bank's database system. When you access a Web site, information about you may be retrieved from a database to select which advertisements you should see. Furthermore, data about your Web accesses may be stored in a database.
Some applications are discussed below:
Telecommunication For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
Airlines For
reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
Enterprise Information • Sales: For the

100% **MATCHING BLOCK 24/202** W

customer, product, and purchase information. • Accounting: For payments, receipts, account balances, assets, and other accounting information. • Human resources: For information about employees, salaries, payroll taxes, and benefits, and for

the generation of paychecks.
14 •
Manufacturing: For the management of the
supply chain and for tracking the

100% **MATCHING BLOCK 25/202** W

production of items in factories, inventories of items in warehouses and stores, and orders for items. • Online retailers: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations. • Banking and Finance • Banking: For customer information, accounts, loans, and banking transactions. • Credit card transactions: For purchases on credit cards and generation of monthly statements. • Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm. Universities For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

The application of a database system can be depicted in the following example for
using the internet. Many of the sites on the Internet are driven by database applications. For example, you may visit an online bookstore that allows you to browse and buy books, such as Amazon.com. The bookstore allows you to browse books in different categories, such as computing or management, or by author name. In either case, there is a database on the organization's Web server that consists of book details, availability, shipping information, stock levels, and order history. Book details include book titles, ISBNs, authors, prices, sales histories, publishers, reviews, and detailed descriptions. The database allows books to be cross-referenced: for example, a book may be listed under several categories,
15 such as computing, programming languages, bestsellers, and recommended titles. The cross-referencing also allows Amazon to give you information on other books that are typically ordered along with the title you are interested in. You can provide your credit card details to purchase one or more books online. Amazon.com personalizes its service for customers who return to its site by keeping a record of all previous transactions, including items purchased, shipping, and credit card details. When you return to the site, you might be greeted by name and presented with a list of recommended titles based on previous purchases. 1.4

Advantages of DBMS The database management system has promising potential advantages. • Control of data redundancy: The traditional file-based systems waste space by storing the same information in more than one file. In contrast, the database approach attempts to eliminate the redundancy by integrating the files so that multiple copies of the same data are not stored. However, the database approach does not eliminate redundancy entirely but controls the amount of redundancy inherent in the database. Sometimes it is necessary to duplicate key data items to model relationships; at other times, it is desirable to duplicate some data items to improve performance. • Data consistency: By eliminating or controlling redundancy, we reduce the risk of inconsistencies occurring. If a data item is stored only once in the database, any update to its value has to be performed only once and the new value is available immediately to all users. If a data item is stored more than once and the system is aware of this, the system can ensure that all copies of the item are kept consistent. Unfortunately, many of today's DBMSs do not automatically insure this type of consistency. • More information from the same data: With the integration of the operational data, it may be possible for the organization to derive additional 16 information from the same data. • Sharing of data: Typically, files are owned by the people or departments that use them. On the other hand, the database belongs to the entire organization and can be shared by all authorized users. In this way, more users share more of the data. Furthermore, new applications can build on the existing data in the database and add only data that is not currently stored, rather than having to define all data requirements again. The new applications can also rely on the functions provided by the DBMS, such as data definition and manipulation, and concurrency and recovery control, rather than having to provide these functions themselves. • Improved data integrity: Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Constraints may apply to data items within a single record or to relationships between records. The integration allows the DBA to define integrity constraints, and the DBMS to enforce them. • Improved security: Database security is the protection of the database from unauthorized users. Without suitable security measures, integration makes the data more vulnerable than file-based systems. However, integration allows the DBA to define database security, and the DBMS to enforce it. This security may take the form of usernames and passwords to identify people authorized to use the database. The access that an authorized user is allowed on the data may be restricted by the operation type (retrieval, insert, update, delete). For example, the DBA has access to all the data in the database; a branch manager may have access to all data that relates to his or her branch office, and a sales assistant may have access to all data relating to properties but no access to sensitive data such as staff salary details. • Enforcement of standards: Again, integration allows the DBA to define 17 and the DBMS to enforce the necessary standards. These may include departmental, organizational, national, or international standards for such things as data formats to facilitate the exchange of data between systems, naming conventions, documentation standards, update procedures, and access rules. • The economy of scale: Combining all the organization's operational data into one database and creating a set of applications that work on this one source of data can result in cost savings. In this case, the budget that would normally be allocated to each department for the development and maintenance of its file-based system can be combined, possibly resulting in a lower total cost, leading to an economy of scale. The combined budget can be used to buy a system configuration that is more suited to the organization's needs. This may consist of one large, powerful computer or a network of smaller computers. •

Balance of conflicting requirements: Each user or department has needs that may be in conflict with the needs of other users. Because the database is under the control of the DBA, the DBA can make decisions about the design and operational use of the database that provide the best use of resources for the organization as a whole. These decisions will provide optimal performance for important applications, possibly at the expense of less-critical ones. • Improved data accessibility and responsiveness: Again, as a result of integration, data that crosses departmental boundaries is directly accessible to the end-users. This provides a system with potentially much more functionality that can, for example, be used to provide better services to the end-user or the organization's clients. Many DBMSs provide query languages or report writers that allow users to ask ad hoc questions and to obtain the required information almost immediately at their terminal, without requiring a programmer to write some software to extract this information from the database.
18 •
Increased productivity: The DBMS provides many of the standard functions that the programmer would normally have to write in a
file-based
application. At a basic level, the DBMS provides all the low-level file- handling routines that are typical in application programs. The provision of these functions allows the programmer to concentrate on the specific functionality required by the users without having to worry about low-level implementation details. Many DBMSs also provide a fourth-generation environment, consisting of tools to simplify the development of database applications. This results in increased programmer productivity and reduced development time (with associated cost savings). • Improved maintenance through data independence: In file-based systems, the descriptions of the data and the logic for accessing the data are built into each application program, making the programs dependent on the data. A change to the structure of the data—such as making an address 41 characters instead of 40 characters, or a change to the way the data is stored on disk—can require substantial alterations to the programs that are affected by the change. In contrast, a DBMS separates the data descriptions from the applications, thereby making applications immune to changes in the data descriptions. This is known as data independence. The provision of data independence simplifies database application maintenance. • Increased concurrency: In some file-based systems, if two or more users are allowed to access the same file simultaneously, it is possible that the accesses will interfere with each other, resulting in loss of information or even loss of integrity. Many DBMSs
manage concurrent
database access and ensure that
such problems cannot occur. •
Improved backup and recovery services:
Many file-based systems place the responsibility on the user to provide measures to protect the data from failures to the computer system or application program. This may involve performing a nightly backup of the data. In the event of a failure during the
19 next day, the backup is restored and the work that has taken place since this backup is lost and has to be re-entered. In contrast, modern DBMSs provide facilities to minimize the amount of processing that is lost following a failure. 1.5 Disadvantages of DBMS
The disadvantages of the database approach are given below: • Complexity: The provision of the functionality that we expect of a good DBMS makes the DBMS an extremely complex piece of software. Database designers and developers, data and database administrators, and end- users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization. • Size: The complexity and breadth of functionality make the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring
substantial amounts of memory to run efficiently. • Cost of DBMSs:
The cost of DBMSs varies significantly, depending on the environment and functionality provided. For example, a single-user DBMS for a personal computer may only cost $100. However, a large mainframe multi-user DBMS servicing hundreds of users can be extremely expensive, perhaps $100,000 or even $1,000,000. There is also the recurrent annual maintenance cost, which is typically a percentage of the list price. • Additional hardware costs: The disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance, it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.
20 •

Cost of conversion: In some situations, the cost of the DBMS and extra hardware may be relatively small compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems, and possibly the employment of specialist staff to help with the conversion and running of the systems. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to more modern database technology. The term legacy system is sometimes used to refer to an older, and usually inferior, system. • Performance: Typically, a file-based system is written for a specific application, such as invoicing. As a result, performance is generally very good. However, the DBMS is written to be more general, to cater to many applications rather than just one. The result is that some applications may not run as fast as they used to. • The greater impact of a failure: The centralization of resources increases the vulnerability of the system. Because all users and applications rely on the availability of the DBMS, the failure of certain components can bring operations to a halt. 1.6 Summary • A database is a shared collection of logically related data and a description of this data, designed to meet the information needs of an organization. • A DBMS is a software system that enables users to define, create, maintain, and control access to the database. • An application program is a computer program that interacts with the database by issuing an appropriate request (typically a SQL statement) to the DBMS. • The DBMS environment consists of hardware (the computer), software (the 21 DBMS, operating system, and applications programs), data, procedures, and people. The people include data and database administrators, database designers, application developers, and end-users. • Some prominent database system applications can be seen in the field of telecommunication, airlines, banking & financial services. • Some advantages of the database approach include control of data redundancy, data consistency, sharing of data, and improved security and integrity. Some disadvantages include complexity, cost, reduced performance, and higher impact of a failure. 1.7 Key Terms • Data Definition Language (DDL): DDL is the language provided by the DBMS to allow users to define the database. • Data Manipulation Language (DML): DML is the language provided by the DBMS to allow users to insert, update, delete, and retrieve data from the database. • Structured Query Language (SQL): It is a typical programming language used to operate databases, especially relational databases. • Entity: An entity is a distinct object (a person, place, thing, concept, or event) in the organization that is to be represented in the database. • Attribute: An attribute is a property that describes some aspect of the object that is to be recorded. • Data Redundancy: It refers to the repetition of the same data at two different places in a common database.

22 1.8 Check Your Progress Short- Answer Type Q1) Define Database management system. Q2) _____ is used to insert, update, delete, and retrieve data from the database.

Q3) Full form of SQL: a) Structured Query Language b) Semi Query Language c) Structured Question Language d) Series Query Language Q4) DDL is the language provided by the DBMS to allow users to define the database. True/ False? Q5) The DBMS is a general-purpose software system that facilitates the processes of: a) Defining b) constructing c) manipulating d) All of the above Long- Answer Type Q1) State at least five advantages and disadvantages of DBMS. Q2) Explain the purpose and significance of DBMS. Q3) Define the following: a) Database b) DDL c) DML Q4) List different types of database users. Q5) Write a short note on components of the DBMS environment.
23 References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
24 Unit: 02 Different types of databases Structure 2.0 Introduction 2.1 Unit Objectives 2.2
Data Models 2.2.1 Object-based Data Models 2.2.2 Record-based Data Models 2.2.3 Physical Data Models 2.2.4 Conceptual Modeling 2.3
Relational Databases 2.4 Distributed Databases 2.5 Centralized Databases 2.6 Difference between Centralized and Distributed Databases 2.7 Summary 2.8 Key Terms 2.9 Check Your Progress 2.0 Introduction The classification of database systems is based on several criteria. The first important criterion

84%	MATCHING BLOCK 27/202	SA	47F417BA15858476660.pdf (D123781188)

is the data model on which DBMS is based. The relational data model

is the most widely used data model
for many commercial DBMSs. Various older applications run on the database systems based on hierarchical and network data models. Another data model that was limited to some commercial systems is the object

79%	MATCHING BLOCK 28/202	SA	47F417BA15858476660.pdf (D123781188)

data model. The relational DBMSs are evolving continuously, and have been incorporating many of the concepts that were developed in object 25 databases. This has led to a new class of DBMSs called object-relational DBMSs. Hence DBMSs can be categorized on the

basis of

97%	MATCHING BLOCK 29/202	SA	47F417BA15858476660.pdf (D123781188)

data models: relational, object, object-relational, hierarchical, network, and other. The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with personal computers. Multiuser systems, which include the majority of DBMSs, support multiple users concurrently. A third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites. A recent trend is to develop software to access several autonomous pre-existing databases stored under heterogeneous llBMSs. This leads to a federated DBMS (or

multi-database

97%	MATCHING BLOCK 30/202	SA	47F417BA15858476660.pdf (D123781188)

system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DBMSs use client- server architecture.

This unit describes various data models and the basic types of databases, like relational, distributed, and centralized databases. 2.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the different types of data models. • Describe the classification of databases. • Illustrate the features of relational, centralized, and distributed databases.
26 2.2 Data Models The
data model is
an integrated collection of concepts for describing and manipulating data, relationships between data, and constraints on the data in an organization. A model is
a representation of real-world objects and events and their associations. It is an abstraction that concentrates on the essential, inherent aspects of an organization and ignores the accidental properties. A data model represents the organization itself. It should provide the basic concepts and notations that will allow database designers and end-users to communicate
unambiguously and accurately their understanding of the organizational data. A data model can be thought of as comprising three components: 1. A structural part, consisting of a set of rules according to which databases can be constructed. 2. A manipulative part, defining the types of operations that are allowed on the data (this includes the operations that are used for updating or retrieving data from the database and for changing the structure of the database). 3. A set of integrity constraints, which ensures that
the data is accurate. The purpose of a data model is to represent data and to make the data understandable.
If it does this, then it can be easily used to design a database.
There have been many data models proposed in the literature. They fall into three broad categories: object-based, record-based, and physical data models. The first two
are used to describe data at the conceptual and external levels, the third is used to describe data at the internal level.
2.2.1
Object-based Data Models Object-based
data models use concepts such as entities, attributes, and relationships. An
entity
is a distinct object (a person, place, thing, concept, event)
in the organization that is to be represented in the database. An
attribute is a
27 property that describes some aspect of the object that we wish to record,
and a relationship is an association between entities. Some of the more common types of
the object-based data model are: •

92% **MATCHING BLOCK 31/202** **W**

Entity-Relationship (ER): The entity-relationship (ER) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design. •

98% **MATCHING BLOCK 32/202** **SA** 47F417BA15858476660.pdf (D123781188)

a collection of basic objects, called entities, and relationships among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects.

Object-oriented:

100% **MATCHING BLOCK 34/202** **W**

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

The ER
model has emerged as one of the main techniques for database design and forms the basis for the database design methodology
used in this book. The object-oriented data model extends the definition of an entity to include not only the attributes that describe the state of the object but also the actions that are associated with the object, that is, its behavior. The object is said to encapsulate both state and behavior. 2.2.2
Record-based Data Models In a record-based model, the database consists of a number of
fixed-format records, possibly of differing types. Each record type defines a fixed number of fields, typically of a fixed length.
There are three principal types of record-based logical data model: the relational
data model, the network data model, and the hierarchical data model. The hierarchical and network data models were developed almost a decade before the relational data model, so their links to
28 traditional file processing concepts are more evident. Relational data model The relational data model is based on the concept of mathematical relations. In the relational model, data and relationships are represented as tables, each of which has a number of columns with a unique name. Figure 2.1 is a sample instance of a relational schema for part of a construction company, showing branch and staff
details.
Figure 2.1 A sample instance of a relational schema
For example, it shows that employee John White is a manager with a salary of £30,000, who works at branch (branchNo) B005, which, from the first table, is at 22 Deer Rd in London. It is important to note that there is a relationship between Staff and Branch: a branch office has staff. However, there is no explicit link between these two tables; it is only by knowing that the attribute branchNo in the Staff relation is the same as the branchNo of the Branch relation that we can establish that a relationship exists.
29
Network data model In the network model, data is represented as collections of records, and relationships are represented by
sets. Compared with the relational model, relationships are explicitly modeled by the sets, which become pointers in the implementation. The records are organized as generalized graph structures with records appearing as nodes (also called segments) and sets as edges in the graph. Figure 2.2 illustrates an instance of a network schema for the same data set presented in Figure 2.1.
Figure 2.2 A sample instance of a network schema
Hierarchical data model The hierarchical model is a restricted type of network model. Again, data is represented as collections of records, and relationships are represented by sets.
However, the hierarchical model allows a node to have only one parent. A hierarchical model can be represented as a tree graph, with records appearing as nodes (also called segments) and sets as edges. Figure 2.3 illustrates an instance of a hierarchical schema for the same data set presented in Figure 2.1.
30
Figure 2.3 A sample instance of a hierarchical schema
Record-based (logical)
data models are used to specify the
overall structure of the database and a higher-level description of the implementation.
Their main drawback
is
that they do not provide adequate facilities for explicitly specifying constraints on the data, whereas the object-based data models lack the means of logical structure specification but provide more semantic substance by allowing the user to specify constraints on the data.

The majority of modern commercial systems are based on the relational paradigm, whereas the early database systems were based on either the network or hierarchical data models. The latter two models require the user to have knowledge of the physical database being accessed, whereas the former provides a substantial amount of data independence. Hence, relational systems adopt a declarative approach to database processing (that is, they specify what data is to be retrieved), but network and hierarchical systems adopt a navigational approach (that is, they specify how the data is to be retrieved). 2.2.3
Physical Data Models
Physical data models describe how data is stored in the computer, representing information such as record structures, record orderings, and access paths.
There are not as many physical data models as logical data models; the most common ones
are
the unifying model
and the frame memory.
31 2.2.4 Conceptual Modeling From an examination of the three-level architecture, we see that the conceptual schema is the heart of the database. It supports all the external views and is, in turn, supported by the internal schema. However, the internal schema is merely the physical implementation of the conceptual schema. The conceptual schema should be a complete and accurate representation of the data requirements of the enterprise (business organizations). If this is not the case, some information about the enterprise will be missing or incorrectly represented and we will have difficulty fully implementing one or more of the external views.
Conceptual modeling or conceptual
database design is the process of constructing a model of the information used in
an enterprise that is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations. This model is called a conceptual data model. Conceptual models are also referred to as "logical models" in the literature. However,
the conceptual model is independent of all implementation details, whereas the logical model assumes knowledge of the underlying data model of the target DBMS. 2.3
Relational Databases
A
data model
is

100% **MATCHING BLOCK 36/202** **SA** 47F417BA15858476660.pdf (D123781188)

a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

It

95% **MATCHING BLOCK 37/202** **W**

is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL. The relational model is

92% **MATCHING BLOCK 45/202** **SA** 47F417BA15858476660.pdf (D123781188)

the relational model and uses a collection of tables to represent both data and the relationships among those data.

an example of a record-based model.
Record-based models are so named
because
the database is structured in fixed-format records of several types.
Each
table contains
records of a particular type. Each record type defines a fixed number of fields or attributes.
The columns of the table correspond to the attributes of the record type.

100% MATCHING BLOCK 38/202 W

A relational database consists of a collection of tables, each of which is assigned 32 a unique name.

For example, consider the instructor table of Figure 2.4 (
a),
which stores information about instructors. The table has four column headers: ID, name, dept name, and salary. Each row of this table records information about an instructor, consisting of the instructor's ID, name, dept_name, and salary. Similarly, the course table of Figure 2.4 (
b)
stores information about courses, consisting of a course_id, title, dept_name, and credits, for each course. Note that each instructor is identified by the value of the column ID, while each course is identified by the value of the column course_id. Figure 2.4 (c) shows a third table, prereq, which stores the prerequisite courses for each course. The table has two columns, course_id, and prereq_id. Each row consists of a pair of course identifiers such that the second course is a prerequisite for the first course. Thus, a row in the prereq table indicates that two courses are related in the sense that one course is a prerequisite for the other. As another example, we consider the table instructor, a row in the table can be thought of as representing the relationship between a specified ID and the corresponding values for name, dept_name, and salary values. (
a) (b)
33 (c) Figure 2.4 (a) The instructor relation (b) The course relation (c)
The prereq

100% MATCHING BLOCK 39/202 W

relation In general, a row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of

the

100% MATCHING BLOCK 40/202 W

table and the mathematical concept of relation, from which the relational data model takes its name. In mathematical terminology, a tuple is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an n-tuple of values, i.e., a tuple with n values, which corresponds to a row in a table.

The
order in which tuples appear in a relation is irrelevant since relation is a set of tuples.
Thus,
in the relational model, the term relation is used to refer to a table, while the term tuple is used to refer to a row. Similarly, the term attribute refers to a column of a table. From Figure 2.4 (
a),
we can see that the relation instructor has four attributes: ID, name, dept name, and salary.
For each attribute of relation,

88% MATCHING BLOCK 41/202 W

there is a set of permitted values, called the domain of that attribute.

Thus, the domain of the salary attribute of the instructor relation is the set of all possible salary values, while the domain of the name attribute is the set of all possible instructor names.

100% MATCHING BLOCK 42/202 W

We require that, for all relations r, the domains of all attributes of r be atomic. A domain is atomic if elements of the domain are considered to be indivisible units. For example, suppose the table instructor

in figure 2.4 (a)

100% MATCHING BLOCK 43/202 W

had an attribute phone_number, which can store a set of phone numbers corresponding to the 34 instructor. Then the domain of phone_number would not be atomic, since an element of the domain is a set of phone numbers, and it has subparts, namely the individual phone numbers in the set.

100% MATCHING BLOCK 44/202 W

The null value is a special value that signifies that the value is unknown or does not exist. 2.4

Distributed Databases
A major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although
we may think that
integration and controlled access implies centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants,
or factories, where each unit maintains its own operational data.
The development of a distributed DBMS that reflects this organizational structure makes the data in all units accessible, and stores data proximate to the location where it is most frequently used,
should improve the ability to share the data and should improve the efficiency with which we can access the data.
A
distributed database is a logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network. Distributed DBMS is
the
software system that permits the management of the distributed database and makes the distribution transparent to users.
A distributed database management system (DDBMS) consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers (replicas) under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing 35 data stored on other computers in the network. Users access the distributed database via applications. Applications are classified as those that do not require data from other sites (local applications) and those that do require data from other sites (global applications). We require DDBMS to have at least one global application. A DDBMS, therefore, has the following characteristics: • a collection of logically related shared data; • data split into a number of fragments; • fragments may be replicated; • fragments/replicas are allocated to sites; • sites are linked by a communications network; • data at each site is under the control of a DBMS; • DBMS at each site can handle local applications, autonomously; • each DBMS participates in at least one global application. It is not necessary for every site in the system to have its own local database, as illustrated by the topology of the DDBMS shown in Figure 2.5.
From the definition of the DDBMS, the system is expected to make the distribution transparent (invisible) to the user. Thus, the fact that a distributed database is split into fragments that can be stored on different computers and perhaps replicated should be hidden from the user. The objective of transparency is to make the distributed system appear like a centralized system. This is sometimes referred to as the fundamental principle of distributed DBMSs. This requirement provides significant functionality for the end-user but, unfortunately, creates many additional problems that have to be handled by the DDBMS.
36
Figure 2.5 Distributed database management system (DDBMS) Distributed processing is

a centralized database that can be accessed over a computer network.
It is important to make a distinction between a distributed DBMS and distributed processing.
The key point with the definition of a distributed DBMS is that the system consists of data that is physically distributed across a number of sites in the network. If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS simply distributed processing. We illustrate the topology of distributed processing in Figure 2.6.
Figure 2.6 Distributed Processing
37 2.5 Centralized Databases Centralized database systems are those that run on a single computer system and do not interact with other computer systems. Such database systems span a range from single-user database systems running on personal computers to high- performance database systems running on high-end server systems. A modern, general-purpose computer system consists of one to a few processors and a number of device controllers that are connected through a common bus that provides access to shared memory as shown in figure 2.7. The processors have local cache memories that store local copies of parts of the memory, to speed up access to data. Each processor may have several independent cores, each of which can execute a separate instruction stream. Each device controller is in charge of a specific type of device (for example, a disk drive, an audio device, or a video display). The processors and the device controllers can execute concurrently, competing for memory access. Cache memory reduces the contention for memory access since it reduces the number of times that the processor needs to access the shared memory. Figure 2.7 Centralized Computer System Computers can be used in two distinct ways: as single-user systems and as multiuser systems. Personal computers and workstations fall into the first category. A typical single-user system is a desktop unit used by a single person, usually with only one processor and one or two hard disks, and usually only one person using the machine at a time. A typical multiuser system, on the other
38 hand, has more disks and more memory and may have multiple processors. It serves a large number of users who are connected to the system remotely. Database systems designed for use by single users usually do not provide many of the facilities that a multiuser database provides. In particular, they may not support concurrency control, which is not required when only a single user can generate updates. Provisions for crash recovery in such systems are either absent or primitive—for example, they may consist of simply making a backup of the database before any update. In contrast, database systems designed for multiuser systems support the full transactional features that we have studied earlier. Although most general-purpose computer systems in use today have multiple processors, they have coarse-granularity parallelism, with only a few processors (about two to four, typically), all sharing the main memory. Databases running on such machines usually do not attempt to partition a single query among the processors; instead, they run each query on a single processor, allowing multiple queries to run concurrently. Thus, such systems support a higher throughput; that is, they allow a greater number of transactions to run per second, although individual transactions do not run any faster. Databases designed for single-processor machines already provide multitasking, allowing multiple processes to run on the same processor in a time-shared manner, giving a view to the user of multiple processes running in parallel. Thus,coarse-granularity parallel machines logically appear to be identical to single-processor machines, and database systems designed for time-shared machines can be easily adapted to run on them. In contrast, machines with fine-granularity parallelism have a large number of processors, and database systems running on such machines attempt to parallelize single tasks (queries, for example) submitted by users. Parallelism is emerging as a critical issue in the future design of database systems. Whereas today those computer systems with multicore processors have only a few cores, future processors will have large numbers of cores. As a result, parallel database
39 systems, which once were specialized systems running on specially designed hardware, will become the norm. Client-Server Systems As personal computers became faster, more powerful, and cheaper, there was a shift away from the centralized system architecture. Personal computers supplanted terminals connected to centralized systems. Correspondingly, personal computers assumed the user-interface functionality that used to be handled directly by the centralized systems. As a result, centralized systems today act as server systems that satisfy requests generated by client systems. Figure 2.8 shows the general structure of a client-server system. Figure 2.8 General structure of a client-server system he functionality provided by database systems can be broadly divided into two parts- the front end and the back end. The back end manages access structures, query evaluation and optimization, concurrency control, and recovery. The front end of a database system consists of tools such as the SQL user interface, forms interfaces, report generation tools, and data mining and analysis tools. The interface between the front end and the back end is through SQL, or through an application program. Certain application programs, such as spreadsheets and statistical-analysis packages, use the client–server interface directly to access data from a back-end server. In effect, they provide front ends specialized for particular tasks. Some transaction-processing systems provide a transactional remote procedure call interface to connect clients with a server. These calls appear like ordinary procedure calls to the programmer, but all the remote procedure calls from a

40 client are enclosed in a single transaction at the server end. Thus, if the transaction aborts, the server can undo the effects of the individual remote procedure calls. 2.6 Difference between Centralized and Distributed Databases We have already learned that a centralized database is basically a type of database that is stored, located as well as maintained at a single location only. This type of database is modified and managed from that location itself. This location is thus mainly any database system or a centralized computer system. The centralized location is accessed via an internet connection (LAN, WAN, etc). This centralized database is mainly used by institutions or organizations. On the other hand, a distributed database is basically a type of database which consists of multiple databases that are connected with each other and are spread across different physical locations. The data that is stored on various physical locations can thus be managed independently of other physical locations. The communication between databases at different physical locations is thus done by a computer network. A comparison between centralized and distributed databases is illustrated in Table 2.1. Table 2.1 Comparison between Centralized and Distributed database Centralized database Distributed database It is a database that is stored, located as well as maintained at a single location only. It is a database that consists of multiple databases that are connected with each other and are spread across different physical locations. This database provides a uniform and Since it is spread across different locations thus
41 complete view to the user. it is difficult to provide a uniform view to the user. The users cannot access the database in case of database failure occurs. In a distributed database if one database fails users have access to other databases. The data access time in the case of multiple users is more in a centralized database. The data access time in the case of multiple users is less in a distributed database. The management, modification, and backup of this database are easier as the entire data is present at the same location. The management, modification, and backup of this database are very difficult as it is spread across different physical locations. This database has more data consistency in comparison to distributed databases. This database may have some data replications thus data consistency is less. A centralized database is less costly. Distributed databases are very expensive. 2.7 Summary • The
data model is
an integrated collection of concepts for describing and manipulating data, relationships between data, and constraints on the data in an organization. •
Object-based
data models use concepts such as entities, attributes, and relationships.
In a record-based model, the database consists of a number of fixed-format records, possibly of differing types.
Physical data models describe how data is stored in the computer, representing information such as record structures, record orderings, and access paths. •

100% **MATCHING BLOCK 46/202** **W**

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. 42 •

92% **MATCHING BLOCK 57/202** **SA** 47F417BA15858476660.pdf (D123781188)

the relational model and uses a collection of tables to represent both data and the relationships among those data. 42 •

A distributed database is a
logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network. •
Centralized database systems are those that run on a single computer system and do not interact with other computer systems. Such database systems span a range from single-user database systems running on personal computers to high-performance database systems running on high-end server systems. 2.8 Key Terms • Conceptual Modeling: It is also known as
conceptual
database design. It
is the process of constructing a model of the information used in
an enterprise that is independent of implementation details. •
Tuple: Tuples are used to store multiple items in a single variable. • Domain:
For each attribute of relation,

88% MATCHING BLOCK 47/202 W

there is a set of permitted values, called the domain of that attribute. •

Granularity: In parallel computing, granularity (or grain size) of a task is a measure of the amount of work (or computation) that is performed by that task. • Coarse-grained parallelism: In coarse-grained parallelism, a program is split into large tasks. Due to this, a large amount of computation takes place in processors. • Fine-grained parallelism: In fine-grained parallelism, a program is broken down into a large number of small tasks. These tasks are assigned individually to many processors.
43 2.9 Check Your Progress Short- Answer Type Q1) Which of the following is an example of an
Object-based logical model? a) Relational model b) Hierarchical model c) Network model d)
Entity-Relationship model Q2) In DBMS, the term used to represent the real world concept or object is known as _____. Q3)
Object-based
data models use concepts such as entities, attributes, and relationships.
True/ False? Q4) A relational database consists of a collection of _____. Q5)

96% MATCHING BLOCK 48/202 W

A domain is atomic if elements of the domain are considered to be _____ units.

Long- Answer Type Q1) Describe the three types of data models briefly. Q2) Differentiate between centralized and distributed databases. Q3) What is a relational database? Also, explain its types. Q4) Explain the client-server system in detail. Q5) Discuss the Entity-Relational data model. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
44
MODULE- II NORMALIZATION & ITS FORMS
45
46
Unit: 03 Normalization Structure 3.0 Introduction 3.1 Unit Objectives 3.2 Purpose of Normalization 3.3 Functional Dependency 3.4 Anomalies in a Database 3.5
The
Normalization Process 3.5.1
First normal form (1NF) 3.5.2 Second normal form (2NF) 3.5.3 Third normal form (3
NF) 3.6
Summary 3.7 Key Terms 3.8 Check Your Progress 3.0 Introduction
When we design a database for an enterprise, the main objective is to create an accurate representation of
the data,
relationships
between the data, and constraints on the data that is pertinent to
the enterprise. To help achieve this objective, we can use one or more database design techniques.
One of the techniques is E-R modeling that we have already discussed. Another database design technique is called normalization. Normalization is a database design technique that
begins by examining the relationships (called functional dependencies) between attributes. Attributes describe some property of the data or of the relationships between the data that is important to the enterprise. Normalization uses a series of tests (described as
47 normal forms) to help identify the optimal grouping for these attributes to ultimately identify a set of suitable relations that supports the data requirements of the enterprise. The main purpose of this unit is to introduce the concept of functional dependencies and describe normalization up to
the Third Normal Form (3NF). 3.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the purpose of normalization techniques. • Gain knowledge to identify functional dependencies for a given relation. • Discuss

the
most commonly used
normal forms:
First Normal Form (1NF), Second Normal Form (2NF),
and Third Normal Form (3NF). 3.2
Purpose of Normalization Normalization is
a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. The purpose of normalization is to identify a suitable set of relations that support the data requirements of an enterprise. The characteristics of a suitable set of relations include the following: • the minimal number of attributes necessary to support the data requirements of the enterprise; • attributes with a close logical relationship (described as
a
functional dependency) are found in the same relation; • minimal redundancy, with each attribute, represented only once, with the important exception of attributes that form all or part of foreign keys, which are essential for the joining of related relations.
48 The benefits of using a database that has a suitable set of relations are that the database will be easier for the user to access and maintain the data, and take up
minimal storage space on the computer.
Normalization is a formal technique that can be used at any stage of database design. However,
here we will
highlight two main approaches for using normalization, as illustrated in Figure 3.1. Approach 1 shows how normalization can be used as a bottom-up standalone database design technique,
and
Approach 2 shows how normalization can be used as a validation technique to check the structure of relations, which may have been created using a top-down approach such as ER modeling. No matter which approach is used, the goal is the same; creating a set of well-designed relations that meet the data requirements of the enterprise. Figure 3.1 shows examples of data sources that can be used for database design. Although the users' requirements specification is the preferred data source, it is possible to design a database based on the information taken directly from other data sources, such as forms and reports.
Figure 3.1 also shows that the same data source can be used for both approaches; however, although this is true in principle, in practice the approach taken is likely to be determined by the size, extent, and complexity of the database being described by the data sources and by the preference and expertise of the database designer. The opportunity to use
normalization as a bottom-up standalone technique (Approach 1) is often limited by the level of detail that the database designer is reasonably expected to manage. However, this limitation is not applicable when normalization is used as a validation technique (Approach 2), as the database designer focuses on only part of the database, such as a single relation, at any one time. Therefore, no matter what the size or complexity of the database, normalization can be usefully applied.
49
Figure 3.1 Normalization used to support database design 3.3 Functional Dependency A functional dependency is an important concept associated with normalization. It describes the relationship between attributes (Maier, 1983).
A
functional dependency is a property of the meaning or
semantics of the attributes in a relation. The semantics indicate how attributes relate to one another and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a constraint between the attributes.
For the
discussion on functional dependencies, assume that a relational schema has attributes (A, B, C, . . . , Z) and that the database is described by a single universal
relation
called R (A, B, C, . . . , Z). This assumption means that every attribute in the database has a unique name.
Functional dependency describes the relationship between attributes in a relation.
For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A ® B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes).
50
Figure 3.2 Schematic of Functional dependency

Consider a relation with attributes A and B, where attribute B is functionally dependent on attribute A. If we know the value of A and we examine the relation that holds this dependency, we find only one value of B in all the tuples that have a given value of A, at any moment in time.
Thus, when two tuples
have the same value of A, they also have the same value
of B.
However, for a given value of B, there may be several different values of A.
The dependency between attributes A and B can be represented diagrammatically, as shown
in
Figure 3.2. An alternative way to describe the relationship between attributes A and B is to say that " A functionally determines B." Some readers may prefer this description, as it more naturally follows the direction of the functional dependency arrow between the attributes. Determinant refers to the
attribute, or group of attributes, on the left-hand side of
the
arrow of a functional dependency. When a functional dependency exists, the attribute or group of attributes on the left-hand side of the arrow
is called the determinant. For example, in Figure 3.2, A is the determinant of B.
An additional characteristic of functional dependencies that is useful for normalization is that their determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right-hand side. This requirement is called full functional dependency. Full functional dependency indicates that if A and B are attributes of a relation, B
is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of
A.
A functional dependency A ® B is
a full functional dependency if removal of any attribute from
A results in the dependency
no longer existing. A functional dependency A ® B is a partial dependency if there is some attribute that can be removed from A and yet the dependency still holds.
51
The functional dependencies that we use in normalization have the following characteristics: • There is a one-to-one relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency. (Note that the relationship in the opposite direction—that is, from the right-hand to the left-hand side attributes—can be a one-to-one relationship or one-to-many relationship.) • They hold for all time. • The determinant has the minimal number of attributes necessary to maintain the dependency with the attribute(s) on the right-hand side. In other words, there must be a full functional dependency between the attribute(s) on the left-hand and right-hand sides of the dependency.
Identifying Functional Dependencies Identifying all functional dependencies between a set of attributes should be quite simple if the meaning of each attribute and the relationships between the attributes are well understood. This type of information may be provided by the enterprise in the form of discussions with users and/or appropriate documentation, such as the users' requirements specification. However, if the users are unavailable for consultation and/or the documentation is incomplete, then—depending on the database application—it may be necessary for the database designer to use their common sense and/or experience to provide the missing information.
We can also
consider the situation where functional dependencies are to be identified in the absence of appropriate information about the meaning of attributes and their relationships. In this case, it may be possible to identify functional dependencies if sample data is available that is a true representation of all possible data values that the database may hold.
52
Identifying the Primary Key for a Relation Using Functional Dependencies
The
main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation. An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation. 3.4
Anomalies in a Database As we already know,

a major aim of relational database design is to group attributes into relations to minimize data redundancy. If this aim is achieved, the potential benefits for the implemented database include the following: • updates to the data stored in the database are achieved with a minimal number of operations, thus reducing the opportunities for data inconsistencies occurring in the database; • reduction in the file storage space required by the base relations thus minimizing costs. Of course, relational databases also rely on the existence of a certain amount of data redundancy. This redundancy is in the form of copies of primary keys (or candidate keys) acting as foreign keys in related relations to enable the modeling of relationships between data.

Relations that have redundant data may have problems called update anomalies, which are classified as insertion, deletion, or modification anomalies.

53 Insertion Anomalies

Figure 3.3 Staff and Branch relations Figure 3.4 StaffBranch Relation

There are two main types of insertion anomaly, which we illustrate using the StaffBranch relation shown in Figure 3.4. • To insert the details of new members of staff into the StaffBranch relation, we must include the details of the branch at which the staff are to be located. For example, to insert the details of new staff located at branch number B007, we must enter the correct details of branch number B007 so that the branch details are consistent with values for branch B007 in other tuples of the StaffBranch relation. The relations shown in Figure 3.3 do not

54 suffer from this potential inconsistency, because we enter only the appropriate branch number for each staff member in the Staff relation. Instead, the details of branch number B007 are recorded in the database as a single tuple in the Branch relation. • To insert details of a new branch that currently has no members of staff into the StaffBranch relation, it is necessary to enter nulls into the attributes for staff, such as staffNo. However, as staffNo is the primary key for the StaffBranch relation, attempting to enter nulls for staffNo violates entity integrity, and is not allowed. We therefore cannot enter a tuple for a new branch into the StaffBranch relation with a null for the staffNo. The design of the relations shown in Figure 3.3 avoids this problem because branch details are entered in the Branch relation separately from the staff details. The details of staff ultimately located at that branch are entered at a later date into the Staff relation. Deletion Anomalies If we delete a tuple from the StaffBranch relation that represents the last member of staff located at a branch, the details about that branch are also lost from the database. For example, if we delete the tuple for staff number SA9 (Mary Howe) from the StaffBranch relation, the details relating to branch number B007 are lost from the database. The design of the relations in Figure 3.3 avoids this problem because branch tuples are stored separately from staff tuples and only the attribute branchNo relates the two relations. If we delete the tuple for staff number SA9 from the Staff relation, the details on branch number B007 remain unaffected in the Branch relation. Modification Anomalies If we want to change the value of one of the attributes of a particular branch in the StaffBranch relation—for example, the address for branch number B003- we must update the tuples of all staff located at that branch. If this modification is not carried out on all the appropriate tuples of the StaffBranch relation, the

55 database will become inconsistent. In this example, branch number B003 may appear to have different addresses in different staff tuples. The previous examples illustrate that the Staff and Branch relations of Figure 3.3 have more desirable properties than the StaffBranch relation of Figure 3.4. This demonstrates that although the StaffBranch relation is subject to update anomalies, we can avoid these anomalies by decomposing the original relation into the Staff and Branch relations. There are two important properties associated with

the

decomposition of a larger relation into smaller relations: • The lossless-join property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations. • The dependency preservation property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations. In other words, we do not need to perform joins on the smaller relations to check whether a constraint on the original relation is violated. 3.5

The Normalization Process We are already aware that

Normalization is a formal technique for analyzing relations based on their primary key (or candidate keys) and functional dependencies (Codd, 1972b). The technique involves a series of rules that can be used to test individual relations so that a database can be normalized to any degree. When a requirement is not met,

the relation violating the requirement must be decomposed into relations that individually meet the

requirements of normalization. Three normal forms were initially proposed called

First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).

Subsequently, R. Boyce and E. F. Codd introduced a stronger definition of

the

third normal form called Boyce– Codd Normal Form (BCNF) (Codd, 1974). With the exception of 1NF, all these

56 normal forms are based on functional dependencies among the attributes of a relation (Maier, 1983). Higher

normal

forms that go beyond BCNF were introduced later such as Fourth Normal Form (4NF) and Fifth Normal Form (5NF) (Fagin, 1977, 1979). However, these later normal forms deal with situations that are very rare.
Normalization is often executed as a series of steps. Each step corresponds to a specific normal form that has known properties. As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies. For the relational data model, it is important to recognize that it is only
the
First Normal Form (1NF) that is critical in creating relations; all subsequent normal forms are optional. However, to avoid the update anomalies, it is generally recommended that we proceed to at least
the Third Normal Form (3NF).
Figure 3.5 provides an overview of the process and highlights the main actions
taken in each step of the process.
Normalization is described
as a bottom-up technique extracting information about attributes from sample forms that are first transformed into
a table
format, which
is described as being in Unnormalized Form (UNF). This table is then subjected progressively to the different requirements associated with each normal form until ultimately the attributes shown in the original sample forms are represented as a set of 3
NF
relations. To simplify the description of normalization we assume that a set of functional dependencies is given for each relation in the worked examples and that each relation has a designated primary key. In other words, it is essential that the meaning of the attributes and their relationships is well understood before beginning the process of normalization. This information is fundamental to normalization and is used to test whether a relation is in a particular normal form.
57
Figure 3.5 The process of Normalization 3.5.1 First Normal Form (1NF) In the first normal form,
we begin the process of normalization by first transferring the data from the source (for example, a standard data entry form) into table format with rows and columns. In this format, the table is in
an
unnormalized Form and is referred to as an unnormalized table. To transform the unnormalized table to First Normal Form, we identify and remove repeating groups within the table. A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table. It should be noted
that in this context, the term “key” refers to the attribute(s) that uniquely identify each row within the
58 unnormalized table. There are two common approaches to removing repeating groups from unnormalized tables: 1. By entering appropriate data in the empty columns or rows containing the repeating data. In other words, we fill in the blanks by duplicating the non- repeating data, where required. This approach is commonly referred to as “flattening” the table. 2.
By placing the repeating data, along with a copy of the original key attribute(s), in a separate relation.
Sometimes the unnormalized table may contain more than one repeating group or repeating groups within repeating groups. In such cases, this approach is applied repeatedly until no repeating groups remain. A set of relations is in 1NF if it contains no repeating groups. For both approaches, the resulting tables are now referred to as 1NF relations containing atomic (or single) values at the intersection of each row and column. Although both approaches are correct, approach 1 introduces more redundancy into the original UNF table as part of the “flattening” process, whereas approach 2 creates two or more relations with less redundancy than in the original UNF table. In other words, approach 2 moves the original UNF table further along the normalization process than approach 1. However, no matter which initial approach is taken, the original UNF table will be normalized into the same set of 3NF relations. 3.5.2
Second Normal Form (2NF) The
Second
Normal Form (2NF) is based on the concept of full functional dependency,
which we
have already discussed. The
second normal form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.
59

The normalization of 1NF relations to 2NF involves the removal of partial dependencies. If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them
in a new relation along with a copy of their determinant. 3.5.3
Third Normal Form (3
NF) Although 2NF relations have less redundancy than those in 1
NF, they may still suffer from update anomalies.
The
normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.
When using the general definitions of 2NF and 3NF, we must be aware of partial and transitive dependencies on all candidate keys and not just the primary key. This can make the process of normalization more complex; however, the general definitions place additional constraints on the relations and may identify hidden redundancy in relations that could be missed. The
trade-off
is whether it is better to keep the process of normalization simpler by examining dependencies on primary keys only, which allows the identification of the most problematic and obvious redundancy in relations or to use the general definitions and increase the opportunity to identify missed redundancy. In fact, it is often the case that
whether we use the definitions based on primary keys or the general definitions of 2NF and 3NF, the decomposition of relations is the same. 3.6
Summary • Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. Normalization is a formal method that can be used to identify relations
based on their keys and the functional dependencies among their attributes.
60 •
One of the main concepts associated with normalization is a
functional dependency, which describes the relationship between attributes in a relation. •
Relations with data redundancy suffer from update anomalies, which can be classified as insertion, deletion, and modification anomalies. •
First Normal Form (1NF) is a relation in which the intersection of each row and column contains one and only one value.
• Second Normal Form (2NF) is
a relation that is in first normal form and every non-primary-
key attribute is fully functionally dependent on the primary key.
Full functional dependency indicates that if A and B are attributes of a relation, B
is fully functionally dependent on A if B is functionally dependent on A but not on any proper subset of
A. • Third Normal Form (3NF) is
a relation that is in first and second normal form in which no non-primary-key attribute is transitively dependent on the primary key.
Transitive dependency is a condition where A, B, and C are attributes of a relation such that if A ® B and B ® C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C). 3.7
Key Terms •
Normalization: A technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. •
Unnormalized Form (UNF): It
is a table that contains one or more repeating groups. •
Determinant:
The determinant of a functional dependency refers to
the attribute, or group of attributes, on the left-hand side of the arrow.
61 •
Update anomalies:
Relations that have redundant data may have problems called update anomalies. •
Partial dependency:
A functional dependency A ® B is a partial dependency if there is some attribute that can be removed from A and yet the dependency still holds. 3.8
Check Your Progress Short- Answer Type Q1) If one attribute is determinant of the second, which in turn is determinant of the
third, then the relation cannot be: a) 1NF b) 2NF c) 3NF d)

None of these Q2) A functional dependency is a relationship between or among the _____. Q3)
First Normal Form (1NF) is a relation in which the intersection of each row and column contains one and only one value.
True/ False? Q4) _____ is
a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.
Q5) Relations with data redundancy suffer from _____ anomalies. a) Forward b) backward c) update d) None of these
Long- Answer Type Q1) Describe the purpose of normalizing data. Q2) Explain the concept of functional dependency.
Q3) Differentiate between ER data modeling and Normalization of database design. Q4) Discuss the various update anomalies in a database system. Q5) Explain the first, second, and third normal forms with examples.
62 References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
63 Unit: 04 Other Normal Forms Structure 4.0 Introduction 4.1 Unit Objectives 4.2 The Boyce-Codd
normal form (BCNF) 4.3 Multi-Valued Dependency (MVD) 4.4 Fourth Normal form (4NF) 4.5 Fifth normal form (5NF) 4.6 Database Design 4.7 Denormalization 4.8 Summary 4.9 Key Terms 4.10 Check Your Progress 4.0 Introduction
In the previous unit, we introduced the technique of normalization and the concept of functional dependencies between attributes. We discussed the benefits of using normalization to support database design and demonstrated how attributes shown on sample forms are transformed into 1
NF, 2NF, and 3
NF relations. In this unit, we consider functional dependencies and describe normal forms that go beyond 3NF such as BCNF, 4NF, and 5NF. Relations in 3NF are normally sufficiently well structured to prevent the problems associated with data redundancy. However, later normal forms were created to identify relatively rare problems with relations that, if not corrected, might result in undesirable data redundancy.
64 4.1
Unit Objectives After completing this unit, the reader will be able to: • Learn about the
inference rules that can identify a set of all functional dependencies for a relation. •
Illustrate the normal forms beyond the
Third Normal Form (3NF), which includes Boyce–Codd Normal Form (BCNF), Fourth Normal Form (4NF),
and Fifth Normal Form (5NF). •
Represent attributes shown on a report as BCNF relations using normalization. •
Explain the concept of multi-valued dependencies and 4NF. 4.2 The Boyce-Codd Normal Form (BCNF) We have already demonstrated how 2NF and 3NF disallow partial and transitive dependencies on the primary key of a relation, respectively. Relations that have these types of dependencies may suffer from the update anomalies
are also been discussed. The
application of the general definitions of 2NF and 3NF may
result in
additional redundancy caused by dependencies that violate one or more candidate keys. However, dependencies can still exist that will cause redundancy to be present in 3NF relations,
despite the additional constraints.
This drawback in 3NF resulted in a stronger normal form called
Boyce–Codd Normal Form.
A relation is considered to be
in BCNF if and only if every determinant is a
candidate key.
To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.
We have already discussed
that a determinant is an attribute, or a group of attributes, on which some other attribute is fully functionally dependent.
The
major
difference between 3NF and BCNF is that for a functional dependency A → B, 3
NF allows this dependency in a relation if
B is a primary-key attribute and A is not a candidate key,
whereas BCNF insists that for this dependency to
65 remain in a relation, A must be a candidate key. Therefore, BCNF is a stronger form of 3NF, such that every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

Violation of BCNF is quite rare, as it may happen only under specific conditions. The potential to violate BCNF may occur when: • the relation contains two (or more) composite candidate keys or • the candidate keys overlap, that is have at least one attribute in common.
Let's consider an example
where
a relation violates BCNF and demonstrate the transformation of this relation to BCNF. This example demonstrates the process of converting a 1NF relation to BCNF relations. In this example, a description of client interviews by members of staff
is given.
The information relating to these interviews is in the ClientInterview relation shown in Figure 4.1. The members of staff involved in interviewing clients are allocated to a specific room on the day of
the
interview. However, a room may be allocated to several members of staff as required throughout a working day. A client is interviewed only once on a given date but may be requested to attend further interviews at later dates.
Figure 4.1 ClientInterview relation
The ClientInterview relation has three candidate keys: (clientNo, interviewDate), (staffNo, interviewDate, interviewTime), and (roomNo, interviewDate, interviewTime). Therefore the ClientInterview relation has three composite candidate keys, which overlap by sharing the
common attribute interviewDate. We select (clientNo, interviewDate) to act as the primary key for this relation. The ClientInterview relation has the following form:
66 ClientInterview (clientNo, interviewDate. interviewTime,
staffNo,
roomNo) The ClientInterview relation has the following functional dependencies: fd1 clientNo,
interviewDate → interviewTime, staffNo, roomNo (Primary key) fd2 staffNo, interviewDate, interviewTime → clientNo (Candidate key)
fd3 roomNo, interviewDate, interviewTime → staffNo, clientNo (Candidate key) fd4 staffNo, interviewDate → roomNo We examine the functional dependencies to determine the normal form of the ClientInterview relation. As functional dependencies fd1, fd2, and fd3 are all candidate keys for this relation, none of these dependencies will cause problems for the relation. The only functional dependency that requires discussion is (staffNo, interviewDate) → roomNo (represented as fd4). Even though (staffNo, interviewDate) is not a candidate key for the ClientInterview relation this functional dependency is allowed in 3NF because roomNo is a primary-key attribute being part of the candidate key (roomNo, interviewDate, interviewTime). As there are no partial or transitive dependencies on the primary key (clientNo, interviewDate), and functional dependency fd4 is allowed, the ClientInterview relation is in 3NF. However, this relation is not in BCNF due to the presence of the (staffNo, interviewDate) determinant, which is not a candidate key for the relation. BCNF requires that all determinants in a relation must be a candidate key for the relation. As a consequence, the ClientInterview relation may suffer from update anomalies. For example, to change the room number for staff number SG5 on the 13-May-14 we must update two tuples. If only one tuple is updated with the new room number, this results in an inconsistent state for the database. To transform the ClientInterview relation to BCNF, we must remove the violating functional dependency by creating two new relations called Interview and
67 StaffRoom, as shown in Figure 4.2. The Interview and StaffRoom relations have the following form: Interview (clientNo, interviewDate, interviewTime, staffNo) StaffRoom (staffNo, interviewDate, roomNo)
Figure 4.2 The Interview and StaffRoom BCNF relations 4.3 Multi-Valued Dependency
Although BCNF removes any anomalies due to functional dependencies, further research led to the identification of another type of dependency called a
Multi- Valued Dependency (MVD), which can also cause data redundancy (Fagin, 1977).
The possible existence of multi-valued dependencies in a relation is due to 1NF, which disallows an attribute in a tuple from having a set of values. For example, if we have two multi-valued attributes in a relation, we have to repeat each value of one of the attributes with every value of the other attribute, to ensure that tuples of the relation are consistent. This type of constraint is referred to as a multi-valued dependency and results in data redundancy. Consider the
68 BranchStaffOwner relation shown in figure 4.3, which
displays the names of members of staff (sName) and property owners (oName) at each branch office (branchNo). In this example, assume that staff name (sName) uniquely identifies each member of staff and that the owner name (oName) uniquely identifies each owner. (
a) (b) Figure 4.3 (a) The BranchStaffOwner relation (b) The BranchStaff and BranchOwner 4NF relations

In this example, members of staff named Ann Beech and David Ford work at branch B003, and property owners named Carol Parrel and Tina Murphy are registered at branch B003. However, as there is no direct relationship between members of staff and property owners at a given branch office, we must create a tuple for every combination of the
member of staff and owner to ensure that the relation is consistent. Multi-Valued Dependency (
MVD)
represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, the set of values for B and C are independent of each other. We represent an MVD between attributes A, B, and C in a relation using the following notation: A —<< B A —<< C
69 A multi-valued dependency constraint potentially exists in the BranchStaffOwner relation,
because two independent relationships are represented in the same relation.
We specify the MVD constraint in the BranchStaffOwner relation shown in Figure 4.3 (a) as follows: branchNo —<< sName branchNo —<< oName A multi-valued dependency can be further defined as being trivial or nontrivial.
An
MVD A—<<B in relation R is defined as being trivial if: (a) B is a subset of A or (b) A > B = R. An MVD is defined as being nontrivial if neither (a) nor (b) are satisfied. A trivial MVD does not specify a constraint on a relation; a nontrivial MVD does specify a constraint.
The BranchStaffOwner relation shown in Figure 4.3(a)
contains two non-trivial dependencies that are
branchNo—<< sName and branchNo—<< oName with branchNo not a candidate key of the
relation. The BranchStaffOwner relation is therefore constrained by the nontrivial MVDs to repeat rows to ensure that the relation remains consistent in terms of the relationship between the sName and oName attributes. For example, if we wanted to add a new property owner for branch B003, we would have to create two new tuples, one for each member of staff, to ensure that the relation remains consistent. This is an example of an update anomaly caused by the presence of the
nontrivial MVDs. We therefore clearly require a normal form that prevents relational structures such as the BranchStaffOwner relation. 4.4
Fourth
Normal Form (4NF) A relation is in 4NF if and only if
for every
non-trivial multi-valued
dependency A—<< B, A is a candidate key of the relation. Fourth normal form (4NF) prevents
70 a relation from containing a non-trivial MVD without the associated determinant being a candidate key for the relation (Fagin, 1977). When the 4NF rule is violated, the potential for data redundancy exists, as shown previously in Figure 4.3 (a). The normalization of a relation breaking the 4NF rule requires
the removal of the offending MVD from the relation by placing the multi-valued attribute(s) in a new relation along with a copy of the determinant. For example, the BranchStaffOwner relation in Figure 4.3 (a) is not in 4NF, because of the presence of two nontrivial MVDs. We decompose the BranchStaffOwner relation into the BranchStaff and BranchOwner relations, as shown in Figure 4.3 (b). Both new relations are in 4NF, because the BranchStaff relation contains the trivial MVD branchNo—<< sName, and the BranchOwner relation contains the trivial MVD branchNo—<< oName. Note that the 4NF relations do not display data redundancy, and the potential for update anomalies is removed. For example, to add a new property owner for branch B003, we simply create a single tuple in the BranchOwner relation. 4.5 Fifth Normal Form (5NF) Whenever we decompose a relation into two relations, the resulting relations have the lossless-join property. This property refers to the ability to rejoin the resulting relations to produce the original relation. However, there are cases
in which there is a requirement
to decompose a relation into more than two relations. Although rare, these cases are managed by join-dependency and Fifth Normal Form (5NF). Let’s briefly discuss the lossless-join dependency and the association with 5NF. • Lossless-Join Dependency:
In splitting relations by projection, we are very explicit about the method of decomposition. In particular, we are careful to use projections that can be reversed by joining the resulting relations, so that the original relation is reconstructed. Such a decomposition is called a lossless-join (also called a
non-loss or non-additive join) decomposition
as
it
preserves all the data in the original relation and does not result in the

71 creation of additional spurious tuples. For example, Figures 4.3 (a) and 4.3 (b) show that the decomposition of the BranchStaffOwner relation into the BranchStaff and BranchOwner relations has the lossless-join property. In other words, the original BranchStaffOwner relation can be reconstructed by performing a natural join operation on the BranchStaff and BranchOwner relations. In this example, the original relation is decomposed into two relations. However, there are cases where we require to perform a lossless- join decompose of a relation into more than two relations (Aho et al., 1979). These cases are the focus of the lossless-join dependency and 5NF. • Definition of Fifth Normal Form: Fifth
normal form (5NF) prevents a relation
from containing a non-trivial join dependency (JD) without the associated projection including a candidate key of the original relation (Fagin, 1977). Non-trivial JDs that are not associated with candidate keys are very rare, so 4NF relations are normally also in 5NF. Although rare, let us examine what potential problems exist for a relation that breaks the rules of 5NF. The PropertyItemSupplier relation shown in Figure 4.4 (a) is not in 5NF, as it contains a nontrivial join dependency constraint. This
relation describes properties (propertyNo) that require certain items (itemDescription), which are supplied by suppliers (supplierNo) to the properties (propertyNo). Furthermore, whenever a property (p) requires a certain item (i), and a supplier (s) supplies that item (i), and the supplier (s) already supplies at least one item to that property (p), then the supplier (s) will also supply the required item (i) to property (p). In this example, assume that a description of an item (itemDescription) uniquely identifies each type of item.
72
Figure 4.4 (a) Illegal state for PropertyItemSupplier relation and (b) legal state for PropertyItemSupplier relation. To identify the type of constraint on the PropertyItemSupplier relation in Figure 4.4 (a), consider the following statement: If Property PG4 requires Bed (from data in tuple 1) Supplier S2 supplies property PG4 (from data in tuple 2) Supplier S2 provides Bed (from data in tuple 3) Then Supplier S2 provides Bed for property PG4 This example illustrates the cyclical nature of the constraint on the PropertyItemSupplier relation. If this constraint holds then the tuple (PG4, Bed, S2) must exist in any legal state of the PropertyItemSupplier relation, as shown in Figure 4.4 (b). This is an example of a type of update anomaly and we say that this relation contains a
non-trivial join dependency (JD) constraint. 4.6
Database Design Database design is one of
the main stages of the database system development lifecycle. This stage starts only after a complete analysis of the enterprise's requirements has been undertaken.
The methodology is presented as a step-by-
73 step guide to the three main phases of database design, namely: conceptual, logical, and physical design. • Conceptual
database design—to build the conceptual representation of the - database, which includes identification of the important entities, relationships, and
attributes.
The conceptual database design phase begins with the creation of a conceptual data model of the enterprise that is entirely independent of implementation details such as the target DBMS, application programs, programming languages, hardware platform, performance issues, or any other physical considerations. • Logical database design—
to translate the conceptual representation to the logical structure of the database, which includes designing the relations.
The logical database design
phase maps the conceptual data
model on to a logical model, which is influenced by the data model for the target database (for example, the relational model).
The logical data model is a source of information for the physical design phase, providing the physical database designer with a vehicle for making trade-offs that are very important to the design of an efficient database. • Physical database design—
to
decide how the logical structure is to be physically implemented (as base relations) in the target
DBMS.
The physical database design phase allows the designer to make decisions on how the database is to be implemented. Therefore, physical design is tailored to a specific DBMS.
There is feedback between physical and logical design, because decisions taken during physical design for improving performance may affect the logical data model.
Before

presenting the methodology, we discuss what a design methodology represents and describe the three phases of database design. Finally, we present guidelines for achieving success in
database
design. A design methodology consists of phases each containing a number of steps that guide the designer in the techniques appropriate at each stage of the project. A
74 design methodology also helps the designer to plan, manage, control, and evaluate database development projects. Furthermore, it is a structured approach for analyzing and modeling a set of requirements for a database in a standardized and organized manner.
The following guidelines are often critical to the success of database design: • Work interactively with the users as much as possible. • Follow a structured methodology throughout the data modeling process. •
Employ a data-driven approach. • Incorporate structural and integrity considerations into the data models. • Combine conceptualization, normalization, and transaction validation techniques into the data modeling methodology. • Use diagrams to represent as much of the data models as possible. • Use a Database Design Language (DBDL) to represent additional data semantics that cannot easily be represented in a diagram. • Build a data dictionary to supplement the data model diagrams and the DBDL. • Be willing to repeat steps.
Table 4.1:
Overview of the Database Design
Methodology Conceptual database design
Logical database design for the relational model Physical database design for relational databases
Step-1: Build
the
conceptual data model. •
Identify entity types. • Identify relationship types. • Identify and associate
Step-2: Build the logical data model. • Derive relations for the
logical data model. • Validate relations
Step-3: Translate logical data model for target DBMS. • Design
base relations. • Design representation of derived data.
75
attributes with entity or relationship types. • Determine attribute domains. • Determine candidate, primary, and alternate key attributes. •
Consider use of enhanced modeling concepts (optional step). • Check model for redundancy. • Validate conceptual data model against user transactions. • Review conceptual data model with
the user.
using normalization. • Validate relations against user transactions. • Check integrity constraints. • Review logical data model with
the
user. • Merge logical data models into global model (optional step). • Check for future growth. •
Design general constraints. Step-4: Design file organizations and indexes. • Analyze transactions. • Choose file organizations. • Choose indexes. • Estimate disk space requirements.
Step-5: Design user views. Step-6: Design security mechanisms.
Step-7: Consider the introduction of controlled redundancy.
Step-8: Monitor and tune the operational system.
Table 4.1 represents an overview of the database design methodology.
This methodology can be used to design relatively simple to highly complex database
systems.
Step-1 creates a conceptual database design, Step-2 creates a logical database design, and Steps-3 to 8 create a physical database design.
An important aspect of any design methodology is to ensure that the models produced are repeatedly validated so that they continue to be an accurate representation of the part of the enterprise being modeled. In this methodology the data models are validated in various ways such as by using normalization, by ensuring the critical transactions are supported, and by involving the users as much as possible. The logical model created at the end of Step 2 is then used as the source of information for physical database design described in Steps 3 to 8. Again,
76 depending on the complexity of the database systems being designed and/or the functionality of the target DBMS, some of
the steps of physical database design may be omitted.
Database design is an iterative process that

has a starting point and an almost endless procession of refinements. Although the steps of the methodology are presented here as a procedural process, it must be emphasized that this does not imply that it should be performed in this manner. It is likely that knowledge gained in one step may alter decisions made in a previous step. Similarly, it may be useful to look briefly at a later step to help with an earlier step. Therefore, the methodology should act as a framework to help guide the designer through database design effectively. 4.7
Denormalization

98% **MATCHING BLOCK 49/202** W

Occasionally database designers choose a schema that has redundant information; that is, it is not normalized. They use the redundancy to improve performance for specific applications. The penalty paid for not using a normalized schema is the extra work (in terms of coding time and execution time) to keep redundant data consistent. For instance, suppose all course prerequisites have to be displayed along with a course information, every time a course is accessed. In our normalized schema, this requires a join of course with prereq. One alternative to computing the join on the fly is to store a relation containing all the attributes of course and prereq. This makes displaying the "full" course information faster. However, the information for a course is repeated for every course prerequisite, and all copies must be updated by the application, whenever a course prerequisite is added or dropped. The process of taking a normalized schema and making it non normalized is called denormalization, and designers use it to tune performance of systems to support time-critical operations. 77

A
better alternative, supported by many database systems today, is to use the normalized schema, and additionally store the join of course and prereq as a materialized view. (A materialized view is a view whose result is stored in the database and brought up to date when the relations used in the view are updated.) Like denormalization, using materialized views does have space and time overhead; however, it has the advantage that keeping the view up to date is the job of the database system, not the application programmer.
Formally, the term denormalization refers to a refinement to the relational schema such that the degree of normalization for a modified relation is less than the degree of at least one of
the original relations. We also use the term more loosely to refer to situations in which we combine two relations into one new
relation, and the new relation is still normalized but contains more nulls than the original relations.
Sometimes denormalization is referred to as usage refinement. 4.8 Summary •
A relation is considered to be
in Boyce-Codd Normal Form (BCNF) if and only if every
determinant is a candidate key. •
A relation is
said to be in
Fourth Normal Form (4NF) if and only if
A is
a candidate key of the relation
for every non trivial multivalued dependency A —<< B. • A multivalued
dependency (MVD) represents a dependency between attributes (A, B, and C) in a relation such that for each value of A there is a set of values of B and a set of values for C. However, the set of values for B and C are independent of each other. • A lossless-join dependency is a property of decomposition, which means that no spurious tuples are generated when relations are combined through a natural join operation.
78 •
A relation is
said to be
in Fifth Normal Form (5NF) if and only if
for every
join dependency (R 1 , R 2 , . . . R n) in a relation R, each projection includes a candidate key of the original relation. •
The database design
methodology is presented as a step-by-step guide to the three main phases of database design, namely: conceptual, logical, and physical design. •

80% MATCHING BLOCK 50/202 W

The process of taking a normalized schema and making it non normalized is called denormalization. 4.9

Key Terms • Multivalued Dependency (MVD): A multivalued dependency (MVD) represents a dependency between attributes in a relation. • Trivial MVD: An
MVD A—<<B in relation R is defined as being trivial if: (a) B is a subset of A or (b) A > B = R. •
Lossless Join Dependency:
It preserves all the data in the original relation and does not result in the creation of additional spurious tuples. •
Database Design: It
is an iterative process that
has a starting point and an almost endless procession of refinements. •
Database Design Language (DBDL): It is used
to represent additional data semantics that cannot easily be represented in a diagram. •
Denormalization: It
refers to a refinement to the relational schema such that the degree of normalization for a modified relation is less than the degree of at least one of the original relations. 4.10
Check Your Progress Short- Answer Type
79 Q1)
Normalization is the process of refining the design of relational tables to minimize data _____. Q2) The Third normal form resolves transitive dependencies. (
True/ False?) Q3) Normal forms are table structures with _____ redundancy. a) Maximum b) Minimum c) Same d) Different Q4) The _____ normal form deals with join-dependencies. a) First b) Second c) Fifth d) Fourth Q5) The fifth form is usually applied only for large relational data models. (
True/ False?) Long- Answer Type Q1) Differentiate
between the Third normal form (3NF) and Boyce Codd Normal Form (
BCNF) with the help of an example. Q2) What is Multivalued dependency? Explain its relationship with 4NF. Q3) Explain how is 5NF related to Join dependencies. Q4) What is Database Design methodology? Also give the overview of the three main phases of database design. Q5) Write a short note on Normalization & Denormalization. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
80
81 MODULE- III TRANSACTION PROCESSING &
DATABASE RECOVERY MANAGEMENT
82
83
Unit: 05 Transaction Processing Structure 5.0 Introduction 5.1 Unit Objectives 5.2 Basics of Transaction Processing 5.3 Serializability and recoverability 5.4 Deadlock Handling 5.4.1 Deadlock Prevention 5.4.2 Deadlock Detection & Recovery 5.5 Multilevel Transaction 5.6 Real- Time Transaction Systems 5.7 Long- Duration Transactions 5.8 Summary 5.9 Key Terms 5.10 Check Your Progress 5.0 Introduction The related functions of a Database management system (
DBMS)
ensure that the database is reliable and remains in a consistent state.
Among these functions, three closely related functions are
transaction support, concurrency control services,
and recovery services. This reliability and consistency must be maintained despite failures of both hardware and software components, and when multiple users are accessing the database. In
the upcoming units we will concentrate on these three functions.
84 The term transaction refers to as a collection

80% MATCHING BLOCK 51/202 W

of operations that form a single logical unit of work on a database.

It may be a single statement, a part of a program, or an entire program.
It may involve any number of operations on the database. Often, the database
user considers a collection of several operations on the database as a single unit. Clearly, it is essential that all these operations occur, or that, in case of a failure, none occur.

100% **MATCHING BLOCK 52/202** W

Furthermore, it must manage concurrent execution of transactions in a way that avoids the introduction of inconsistency.

This unit introduces the basic concepts of transaction processing. 5.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the basics and importance of transaction processing. • Illustrate the different properties of transactions. • Discuss the meaning of serializability and recoverability. • Gain knowledge about Deadlock handling in databases. • Describe the other important concepts of transaction systems. 5.2 Basics

85% **MATCHING BLOCK 53/202** W

of Transaction Processing A transaction is defined as a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (like C++, or Java).

88% **MATCHING BLOCK 54/202** W

The transaction consists of all operations executed between the begin transaction and end transaction. This collection of steps must appear to the user as a single, indivisible unit. If a transaction

100% **MATCHING BLOCK 55/202** W

begins to execute but fails for whatever reason, any changes to the database that the transaction may have made must be undone.

This requirement
85 holds regardless of whether the transaction itself failed, the operating system crashed, or the computer itself stopped operating.
A transaction can have one of two outcomes. If it completes successfully, the transaction is said to have committed and the database reaches a new consistent state. On the other hand, if the transaction does not execute successfully, the transaction is aborted. If a
transaction
is aborted,
the database must be restored to the consistent state it was in before the transaction
started. Such a transaction is rolled back or undone.
It should be noted that a committed transaction cannot be aborted. If the committed transaction is considered as a mistake, then another compensating transaction is performed to reverse the effects.
However, an
aborted transaction that is rolled back can be restarted later
and, depending on the cause of the failure, may successfully execute and commit at that time.
Figure 5.1 State Transition diagram for a transaction Figure 5.1 represents the state transition diagram for
a transaction process.
The keywords BEGIN_TRANSACTION, COMMIT, and ROLLBACK are available in many data manipulation languages to delimit transactions. If these delimiters are not used, the entire program is usually regarded as a single transaction, with the
database system
automatically performing a COMMIT when the program terminates correctly and a ROLLBACK if it does not.
It should be noted

that in addition to the obvious states of ACTIVE, COMMITTED, and ABORTED, there are two other states: • PARTIALLY COMMITTED:
This state occurs on execution of
final statement. At this point, it may be found that the transaction has violated serializability or has violated an integrity constraint and the transaction
86 has to be aborted. Alternatively, the system may fail and any data updated by the transaction may not have been safely recorded on secondary storage. In such cases, the transaction would go into the FAILED state and would have to be aborted. If the transaction has been successful, any updates can be safely recorded and the transaction can go to the COMMITTED state. • FAILED: This state occurs if the transaction cannot be committed or the transaction is aborted while in the ACTIVE state, perhaps due to the user aborting the
transaction or as a result of the concurrency control protocol aborting the transaction to ensure serializability. Properties of Transactions:
The database system should maintain the following properties of the transactions: • Atomicity: It is also known as " all or nothing" property. A transaction is an indivisible unit that is either performed in its entirety or is not performed at all. It is the responsibility of the recovery subsystem of the database system to ensure
the atomicity property of
the transaction. •
Consistency: A transaction must
transform
the database from one consistent state to another consistent state.
Both the
database systems and the application developers
should be responsible to ensure the consistency of the transaction.
The consistency can be ensured by enforcing all
the constraints that have been specified on the database schema, such as integrity
constraints. However,
it is insufficient to ensure consistency
in itself.
For example,
a transaction is initiated to transfer money from one bank account to another and the programmer makes an error in the transaction logic. One account
is debited but the wrong account is credited;
then the database is in an inconsistent state. However, the DBMS would not
be able to detect the error and
the inconsistency.
87 •
Isolation: Transactions execute independently of one another. In other words, the
partial effects of incomplete transactions should not be visible to other transactions.
It is the responsibility of the concurrency control subsystem to ensure isolation. • Durability:
The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost
because of a subsequent failure. It is the responsibility of the recovery subsystem to
ensure
the durability of the transaction. These properties are often called ACID Properties; where ACID represents Atomicity, Consistency, Isolation, and Durability. Figure 5.2 DBMS transaction Subsystem A DBMS transaction subsystem is shown in figure 5.2 that depicts handling of
transactions, concurrency control, and recovery. The transaction manager coordinates transactions on behalf of application programs. It communicates with the scheduler, the module responsible for implementing a particular strategy for concurrency control. The scheduler is also referred to as the lock manager if the concurrency control protocol is locking-based. The objective of the scheduler is to maximize concurrency without allowing concurrently executing transactions to interfere with one another, and so compromise the integrity or consistency of the database.
88 If a failure occurs during the transaction, then the database could be inconsistent. It is the task of the recovery manager to ensure that the database is restored to the state it was in before the start of the transaction, and therefore a consistent state. Finally, the buffer manager is responsible for the efficient transfer of data between disk storage and main memory. 5.3
Serializability and recoverability

The
process of managing simultaneous operations on the database without having them interfere with one another is called concurrency control.
The
objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference between them.
One transaction should be executed
at a time i.e. one transaction is committed before the next transaction is allowed to begin. However, the aim of a multi-user DBMS is also to maximize the degree of concurrency or parallelism in the system, so that transactions that can execute without interfering with one another can run in parallel. For example, transactions that access different parts of the database can be scheduled together without interference.
Serializability can be examined
as a means of helping to identify those executions of transactions that are guaranteed to ensure consistency.
Some important related definitions are given below: • Schedule: It is defined as
a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions. A transaction comprises a sequence of operations consisting of read and/or write actions to the database, followed by a commit or abort action. A schedule S consists of a sequence of the operations from a set of n transactions T 1 , T 2 , . . . , T n , subject to the constraint that the order of operations for each transaction is preserved in the schedule. Thus, for each
transaction
T i in schedule S, the order of the operations in T i must be the same in schedule S.
89 • Serial Schedule:
A
schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions,
is known as serial schedule.
In
a serial schedule, the transactions are performed in serial order. For example, if we have two transactions T 1
and
T 2 , serial order would be T 1 followed by T 2 , or T 2 followed by T 1 . Thus, in serial execution there is no interference between transactions, because only one is executing at any given time. However, there is no guarantee that the results of all serial executions of a given set of transactions will be identical. •
Non Serial
Schedule:
A schedule where the operations from
a set of concurrent transactions are interleaved,
is known as non
serial schedule. Serial execution never leaves the database in an inconsistent state, so every serial execution is considered correct, although different results may be produced. The
objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another,
and thereby produce a database state that could be produced by a serial execution. If a set of transactions execute concurrently, the (nonserial) schedule is correct if it produces the same result as some serial execution. Such a schedule is called serializable. To prevent inconsistency from transactions interfering with one another, it is essential to guarantee serializability of concurrent transactions.
In serializability, the ordering of read and write operations is important: ? If two transactions only read a data item, they do not conflict and order is not important. ? If two transactions either read or write completely separate data items, they do not conflict and order is not important. ? If one transaction writes a data item and another either reads or writes the same data item,
the order of execution is important.
90 Let's consider schedule S 1 in Figure 5.3 (a) having operations from two concurrently executing transactions T 7 and T 8 . Because the write operation on bal x in T 8 does not conflict with the subsequent read operation on bal y in T 7 , the order of these operations
can be changed
to produce the equivalent schedule S 2 shown in Figure 5.3 (b). Now, if we change the order of the following non-conflicting operations, the equivalent serial schedule S 3

is produced as shown in Figure 5.3 (c): Conflict Serializability ?
Change the order of the write (bal x) of T 8 with the write (bal y)
of T 7 . ? Change the order of the read (bal x) of T 8 with the read (bal y) of T 7 . ? Change the order of the read (bal x) of T 8 with the write (bal y) of T 7 .
Figure 5.3 (a) Non Serial
schedule S 1 ; (b) nonserial schedule S 2 equivalent to S 1 ; (
c) serial schedule S 3 , equivalent to S 1 and S 2 .
To test the conflict serializability, consider
the constrained write rule, stating that
a transaction updates a data item based on its old value, which is first read
by the transaction; a precedence (or serialization) graph can be produced to test for conflict serializability. For a schedule S, 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 E, which is constructed as follows: 1. Create a node for each transaction. 2. Create
a directed edge
T i →T j , if T j
reads the value of an item written by T
i . 3. Create
a directed edge
T i →T j , if T j
writes a value into an item after it has been read by
T i . 4. Create a directed edge T i →T j , if T
j
writes a value into an item after it has been written by T i . If
an edge
T i →T j
exists in the precedence graph for S,
then in any serial schedule S' equivalent to S, T i must appear
beforeT j . If the precedence graph contains a cycle, the schedule is not conflict serializable.
Let us consider an example with two transactions T 9 & T 10 . In figure 5.4 (a), it can be seen that
T 9 is transferring £100 from one account with balance bal x to another account with balance bal y. T 10 is increasing the balance of these two accounts by 10%. The precedence graph for this schedule, shown in Figure 5.4 (
b), has a cycle and so is not conflict serializable. (a) (b)
92 Figure 5.4 (a) Two non- conflict serializable transactions (b) Precedence graph
There are several other types of serializability that offer less rigid definitions of schedule equivalence than that offered by conflict serializability. One less restrictive definition is called view serializability. View Serializability Two schedules S 1 and S 2 consist of the same operations from n transactions T 1 , T 2 , . . . , T n are view equivalent if the following three conditions hold: 1. For each data item x, if transaction
T i
reads the initial value of x in schedule S 1 , then transaction T i must also read the initial value of x in schedule S 2 . 2. For each read operation on data item x by transaction T i in schedule S 1 , if the value read by T i has been written by transaction T
j ,
then transaction T i must also read the value of x produced by transaction
T
j
in schedule S 2 . 3. For each data item x , if
the last write operation on x was performed by transaction T i in schedule S 1 , the
same transaction must perform the final write on data item x in
schedule
S 2 .
A schedule is
said to be
view serializable, if it is
view equivalent to a serial schedule.
Every conflict serializable schedule is view serializable, although the converse is not true. Testing for

view serializability is much more complex than testing for conflict
serializability.
To test for view serializability we need a method to decide whether an edge should be inserted into the precedence graph. The approach taken to construct a labeled precedence graph for the schedule
is
as follows: 1. Create a node for each transaction. 2. Create a node labeled
T
bw
which
is a dummy transaction inserted at the beginning of the schedule containing a write operation for each data item accessed in the schedule.
93 3.
Create a node labeled T
fr which denotes
a dummy transaction added at the end of the schedule containing a read operation for each data item accessed in the schedule. 4.
Create
a directed edge T i T j if T j ,
reads the value of an item written by T
i . 5. Remove all directed edges incident on transaction T i for which there is no path from
T i to T fr . 6. For each data item that T
j reads that
has been written by T i , and T k writes (T k ≠ T bw), then: a) If T i = T bw and T j ≠ T fr , then create a directed edge T i T k
. b) If T i ≠ T bw and T j = T fr ,
then create a directed edge T k T i . c) If T i ≠ T bw and T j ≠ T fr , then create a pair of directed edge T k T i , and T j T k ,
where x is a unique positive integer that has not been used for labeling an earlier directed edge. This rule is a more general case of the preceding two rules, indicating that if transaction T i writes an item that T
j subsequently reads, then any transaction,
T k ,
that writes the same item must either precede
T i or succeed T
j . Based on the
labeled precedence graph, the test for view serializability is as follows: • If the graph contains no cycles, the schedule is view serializable. • The presence of a cycle, however, is not a sufficient condition to conclude that the schedule is not view serializable. The actual test is based on the observation that rule 6(c) generates m distinct directed edge pairs, resulting in 2 m different graphs containing just one edge from each pair. If any one of these graphs is acyclic, then the corresponding schedule is view serializable and the serializability order is determined by the topological sorting of the graph with the dummy transactions
T
bw and T fr removed.
94
Serializability identifies schedules that maintain the consistency of the database, assuming that none of the transactions in the schedule fails. An alternative perspective examines the recoverability of transactions within a schedule. If a transaction fails the atomicity property
and requires
the effects of the transaction to be undone. Also, the durability property states that once a
transaction commits, its changes cannot be undone (without running another, compensating, transaction).
Recoverability
Recoverable schedule is
a schedule in which for each pair of transactions T i and T j , if
T j reads
a data item previously written
by T i ,
then the commit operation of T i precedes the commit operation of T
j .
Serializability can be achieved

using the
two main concurrency control techniques
that allow transactions to execute safely in parallel subject to certain constraints are called locking and timestamping. 5.4
Deadlock Handling A system is said to be

83%	MATCHING BLOCK 56/202	W

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction. More precisely, there exists a set of waiting transactions {T 0 , T 1 , . . . , T n } such that T 0 is waiting for a data item that T 1 holds, and T 1 is waiting for a data item that T 2 holds, and T n-1 is waiting for a data item that T n holds, and T n is waiting for a data item that T 0 holds. None of the transactions can make progress in such a situation.

88%	MATCHING BLOCK 58/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction. More precisely, there exists a set of waiting transactions {T 0 , T 1 , . . . , T n } such that T 0 is waiting for a data item that T 1 holds, and T 1 is waiting for a data item that T 2 holds, and T n-1 is waiting for a data item that T n holds, and T n is waiting for a data item that T 0 holds. None of the transactions can make progress in such a situation. The only solution to this undesirable situation is for the system to invoke some drastic action, such as rolling back some of the transactions involved in the deadlock. Rollback of a transaction may be partial: That is, a transaction may be rolled back to the

particular

100%	MATCHING BLOCK 59/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

point where it obtained a lock whose release resolves the deadlock.

There are three general techniques for handling deadlock: timeouts, deadlock
95 prevention, and deadlock detection and recovery. With timeouts, the
transaction that has requested a lock waits for at most a specified period of time.
Using deadlock prevention, the DBMS looks ahead to determine if a transaction would cause deadlock, and never allows deadlock
to occur. Using deadlock detection and recovery, the DBMS allows deadlock to occur but recognizes occurrences of deadlock and breaks them. Because it is more difficult to prevent deadlock than to use timeouts or test for deadlock and break it when it occurs, systems generally avoid the deadlock prevention method.
It should also be noted that the

94%	MATCHING BLOCK 60/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

prevention method is commonly used if the probability that the system would enter a deadlock state is relatively high; otherwise, detection and recovery are more efficient. 5.4.1

Deadlock Prevention

95% **MATCHING BLOCK 61/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Deadlock prevention offers two approaches to proceed. • First approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together. The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock, whenever the wait could potentially result in a deadlock. The simplest scheme under the first approach requires that each transaction locks all its data items before it begins execution. Moreover, either all are locked in one step or none are locked. There are two main disadvantages to this protocol: (1) it is often hard to predict, before the transaction begins, what data items need to be locked; (2) data-item utilization may be very low, since many of the data items may be locked but unused for a long time. • Second approach for preventing deadlocks is to impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering.

The approach uses

59% **MATCHING BLOCK 62/202** SA DBMS_AIML_FINAL.pdf (D111167082)

preemption and transaction rollbacks. In preemption, when a transaction T j requests a lock that transaction T i holds, the lock granted to T i may be 96 preempted by rolling back of T i , and granting of the lock to

T j .

89% **MATCHING BLOCK 63/202** SA DBMS_AIML_FINAL.pdf (D111167082)

If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed: (i) The wait–die scheme is a nonpreemptive technique. When transaction T i requests a data item currently held by

76% **MATCHING BLOCK 64/202** W

If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed: (i) The wait–die scheme is a nonpreemptive technique. When transaction T i requests a data item currently held by T j , T i is allowed to wait only if it has a timestamp smaller than that of T j (that is, T i is older than

79% **MATCHING BLOCK 65/202** SA DBMS_AIML_FINAL.pdf (D111167082)

is allowed to wait only if it has a timestamp smaller than that of T j (that is, T i is older than

T j). Otherwise, T i

66% **MATCHING BLOCK 67/202** W

is rolled back (dies). (ii) The wound–wait scheme is a preemptive technique. It is opposite to the wait–die scheme. When transaction T i requests a data item currently held by T j , T i is allowed to wait only if it has a timestamp larger than that of T j (that is, T i is younger than T j). Otherwise, T j is rolled back (T j is wounded by

88% **MATCHING BLOCK 66/202** SA DBMS_AIML_FINAL.pdf (D111167082)

The wound–wait scheme is a preemptive technique. It is opposite to the wait–die scheme. When transaction

is allowed to wait only if it has a timestamp larger than that of T j (that is, T i is younger than

T i). Timeouts A simple and practical
approach to deadlock handling is based on lock timeouts. In
this approach, a transaction that requests a lock will wait for only a system- defined period of time. If the lock has not been granted within
this period, the lock request times out. In this case, the DBMS assumes that the transaction may be deadlocked, even though it may not be, and it aborts and automatically restarts the transaction. 5.4.2
Deadlock Detection & Recovery Deadlock detection & recovery scheme is used when a system does not employ some protocol that ensures deadlock freedom. Occurrence of deadlock is determined periodically with the help of an algorithm that examines the state of the system. If a deadlock has occurred, then the system must attempt to recover from the deadlock. To recover the system, it must: ? Maintain information about the current allocation of data items to transactions, as well as any outstanding data item requests.
97 ? Provide an algorithm that uses this information to determine whether the system has entered a deadlock state. ? Recover from the deadlock when the detection algorithm determines that a deadlock exists.
Deadlock Detection Deadlock detection is usually handled by the construction of a wait-for graph (WFG) that shows the transaction dependencies; that is, transaction T i is dependent on T j if transaction T j holds the lock on a data item that T i is waiting for. The WFG is a directed graph G = (N, E) that consists of a set of nodes N and a set of directed edges E, which is constructed as follows: ?
Create a node for each transaction. ? Create a directed edge
T i →T j ,
if
transaction T i is waiting to lock an item that is currently locked by
T
j . Figure 5.5 shows a wait-for graph with a cycle showing deadlock between two transactions T 4 & T 5 . It should be noted that
deadlock exists if and only if the WFG contains a cycle.
Because
a cycle in the wait-for graph is a necessary and sufficient condition for deadlock to exist, the deadlock detection algorithm generates the WFG at regular intervals and examines it for a cycle.
It is important to choose
the choice time interval between executions of the algorithm. If the interval chosen is too small, deadlock detection will add considerable overhead; if the interval is too large, deadlock may not be detected for a long period. Alternatively, a dynamic deadlock detection algorithm could start with an initial interval size. Each time no deadlock is detected, the detection interval could be increased.
98
Figure 5.5 WFG with a cycle showing deadlock between two transactions
Recovery from Deadlock Detection We already know that
once deadlock has been detected the DBMS needs to abort one or more of the transactions. There are several issues that need to be considered: 1. Choice of deadlock victim: •
For
how long the transaction has been running. It may be better to abort a transaction that has just initiated rather than one that has been running for
a while.
In some situations, the choice of transactions to abort may be obvious. However, in case the choice is not
clear, there will be a need to abort the transactions that suffer the minimum costs. Following points should be considered to make the right choice: •
How many data items have been updated by the transaction. It would be better to abort a transaction that has made little change to the database rather than one that has made significant changes to the database. • How many data items the transaction is still to update. It would be better to abort a transaction that has many changes still to make to the database rather than one that has few changes to make. Unfortunately, this may not be something that the DBMS would necessarily know. 2. How far to roll a transaction back:
If it is decided to abort a particular transaction, It has to be decided

how far to roll the transaction back. Clearly, undoing all the changes made by a transaction is the simplest solution, although not necessarily the most efficient. It may be possible to resolve the
99 deadlock by rolling back only part of the transaction. 3. Avoiding starvation: Starvation occurs when the same transaction is always chosen as the victim, and the transaction can never complete.
The DBMS can avoid starvation by storing a count of the number of times a transaction has been selected as the victim and using a different selection criterion once this count reaches some upper limit. 5.5
Multilevel Transaction A long-duration transaction can be viewed as a collection of related subtasks or subtransactions. Parallelism can be enhanced by structuring a transaction as a set of subtransactions, since it may be possible to run several subtransactions in parallel. Furthermore, it is possible to deal with failure of a subtransaction (due to abort, system crash, and so on) without having to roll back the entire long- duration transaction. A nested or multilevel transaction T consists of a set T = {t 1 , t 2 , . . . , t n } of subtransactions and a partial order P on T. A subtransaction t i in T may abort without forcing T to abort. Instead, T may either restart t i or simply choose not to run t i . If t i commits, this action does not make t i permanent. Instead, t i commits to T, and may still abort if T aborts. An execution of T must not violate the partial order P. That is, if an edge t i → t j appears in the precedence graph, then t j → t i must not be in the transitive closure of P. Nesting may be several levels deep, representing a subdivision of a transaction into subtasks, sub-subtasks, and so on. At the lowest level of nesting, we have the standard database operations read and write that we have used previously. If a subtransaction of T is permitted to release locks on completion, T is called a multilevel transaction. When a multilevel transaction represents a long duration activity, the transaction is sometimes referred to as a saga. Alternatively, if locks held by a subtransaction t i of T are automatically assigned to T on completion of t i , T is called a nested transaction.
100 5.6 Real- Time Transaction Systems In certain applications, the constraints in DBMS include deadlines by which a task must be completed. For example, traffic control, plant management, and scheduling. When deadlines are included, database consistency is not only concerned about accuracy of an execution; rather, it is concerned with how many deadlines are missed, and by how much time they are missed. These systems with deadlines are called real-time systems. Deadlines are characterized as follows: • Hard deadline: If the task is not completed by the deadline, serious problems, like system crash, may occur. • Firm deadline: If the task is completed after the deadline, it has zero value. • Soft deadline: The task has diminishing value if it is completed after the deadline, with the value approaching zero as the degree of lateness increases. It is essential for transaction management in real-time systems to consider deadlines into account. If it is determined from the concurrency-control protocol that a transaction T i must wait, it may cause T i to miss the deadline. In such cases, it may be preferable to pre-empt the transaction holding the lock, and to allow T i to proceed. Pre-emption must be used with care, however, because the time lost by the preempted transaction (due to rollback and restart) may cause the pre-empted transaction to miss its deadline. Unfortunately, it is difficult to determine whether rollback or waiting is preferable in a given situation. A major difficulty in supporting real-time constraints arises from the variance in transaction execution time. In the best case, all data accesses reference data in the database buffer. In the worst case, each access causes a buffer page to be written to disk (preceded by the requisite log records), followed by the reading from disk of the page containing the data to be accessed. Because the two or more disk accesses required in the worst case take several orders of magnitude
101 more time than the main-memory references required in the best case, transaction execution time can be estimated only very poorly if data is resident on disk. Hence, main-memory databases are often used if real-time constraints have to be met. In real-time systems, deadlines, rather than absolute speed, are the most important issue. Designing a real-time system involves ensuring that there is enough processing power to meet deadlines without requiring excessive hardware resources. Achieving this objective, despite the variance in execution time resulting from transaction management, remains a challenging issue. 5.7 Long- Duration Transactions Initially, the transaction concept is developed according to the data-processing applications. In such applications, most transactions are non-interactive and of short duration. Serious problems arise when this concept is applied to database systems that involve human interaction. Such long duration transactions have the following key properties: • Long duration: Once a human interacts with an active transaction, that transaction becomes a long-duration transaction from the perspective of the computer, since human response time is slow relative to computer speed. Furthermore, in design applications, the human activity may involve hours, days, or an even longer period. Thus, transactions may be of long duration in human terms, as well as in machine terms. • Exposure of uncommitted data: Data generated and displayed to a user by a long-duration transaction are uncommitted, since the transaction may abort. Thus, users and other transactions may be forced to read uncommitted data. If several users are cooperating on a project, user transactions may need to exchange data prior to transaction commit. • Subtasks: An interactive transaction may consist of a set of subtasks
102 initiated by the user. The user may wish to abort a subtask without necessarily causing the entire transaction to abort. • Recoverability: It is unacceptable to abort a long-duration interactive transaction because of a system crash. The active transaction must

be recovered to a state that existed shortly before the crash
so that relatively little human work is lost. • Performance: Good performance in an interactive transaction system is defined as fast response time. This definition is in contrast to that in a non- interactive system, in which high throughput (number of transactions per second) is the goal. Systems with high throughput make efficient use of system resources. However, in the case of interactive transactions, the most costly resource is the user. If the efficiency and satisfaction of the user is to be optimized, response time should be fast (from a human perspective). In those cases where a task takes a long time, response time should be predictable (that is, the variance in response times should be low), so that users can manage their time well. 5.8 Summary •

83%	MATCHING BLOCK 68/202	W

A transaction is defined as a unit of program execution that accesses and possibly updates various data items. •

Atomicity, Consistency, Isolation, and Durability (ACID) are the properties of transactions that a DBMS should maintain. •

100%	MATCHING BLOCK 69/202	W

Atomicity ensures that either all the effects of a transaction are reflected in the database, or none are; a failure cannot leave the database in a state where a transaction is partially executed. • Consistency ensures that, if the database is initially consistent, the execution of the transaction (by itself) leaves the database in a consistent state. 103 • Isolation ensures that concurrently executing transactions are isolated from one another, so that each has the impression that no other transaction is executing concurrently with it. • Durability ensures that, once a transaction has been committed, that transaction's updates do not get lost, even if there is a system failure. •

If a set of transactions execute concurrently, the (nonserial) schedule is correct if it produces the same result as some serial execution. Such a schedule is called serializable. •
If a transaction fails the atomicity property and requires the effects of the transaction
to be undone, then it can be achieved using recoverability. • A system is said to be

92%	MATCHING BLOCK 71/202	W

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction.

92%	MATCHING BLOCK 72/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction.

There are three general techniques for handling deadlock: timeouts, deadlock prevention, and deadlock detection and recovery. •
The systems with deadlines are called real-time systems. A long-duration transaction can be viewed as a collection of related subtasks or subtransactions. 5.9 Key Terms • Rolled back transaction:
If a transaction
is aborted,
the database must be restored to the consistent state it was in before the transaction
started. Such a transaction is rolled back or undone. •
Schedule: It is defined as
a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions. • Multilevel Transaction: If a subtransaction of
T is permitted to release locks on completion, T is called a multilevel transaction.
104 • Saga: When a multilevel transaction represents a long duration activity, the transaction is sometimes referred to as a saga. • Nested Transaction: If locks held by a subtransaction t i of T are automatically assigned to T on completion of t i , T is called a nested transaction. 5.10 Check Your Progress Short- Answer Type Q1)
Which of the following is not a transaction state? a) Active b) Compensated c) Failed d)

Partially committed Q2) _____ is commonly known as the “all-or-none” property of transaction. Q3) A transaction that has not been completed successfully is known as a partially committed transaction. (True/ False?) Q4) When more than one user is accessing the same data at the same time then it is known as _____ . Q5) The systems with deadlines are called real-time systems. (True/ False?) Long- Answer Type Q1) Distinguish between serial schedule and serializable schedule. Q2) Explain the essential properties of Transactions. Q3) Write a short note on Deadlock handling. Briefly explain the general techniques for handling the deadlocks. Q4) What are the two approaches for deadlock prevention? Explain briefly. Q5) What are the key properties of Long- duration transactions?
105 References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
106 Unit: 06 Database Recovery Management Structure 6.0 Introduction 6.1 Unit Objectives 6.2 Need for Recovery 6.3 Recovery & Transactions 6.4 Provisions for Recovery 6.5 Failure Classification 6.6 Recovery Techniques 6.7 Recovery in Multidatabase Systems 6.8 Remote Backup Systems 6.9 Summary 6.10 Key Terms 6.11 Check Your Progress 6.0 Introduction There are different causes for a computer system to fail or crash. It can be a disk crash, power outage, software error, or even sabotage. Any kind of failure will ultimately result in loss of information.

95% **MATCHING BLOCK 75/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Therefore, the DBMS must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is a recovery system that can restore the database to the consistent state that existed before the failure. The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a failure.

The presence of database recovery, provided by
the DBMS, ensures the reliability and consistency of the database in the presence of
failures. This unit examines
107
the need for recovery and the types of failure that can occur in a database environment.
The
concepts and techniques that can minimize these effects and allow recovery from failure
are also discussed. 6.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the need for recovery in databases. • Discuss the classification of failures. • Describe the remote backup systems. • Illustrate the different recovery techniques. 6.2
Need for Recovery The storage of data generally includes four different types of media with an increasing degree of reliability: main memory, magnetic disk, magnetic tape, and optical disk. Main memory is volatile storage that usually does not survive system crashes. Magnetic disks provide online nonvolatile storage. Compared with main memory, disks are more
reliable and much cheaper, but slower
in efficiency.
Magnetic tape is an offline nonvolatile storage medium, which is far more reliable than disk and fairly inexpensive, but slower, providing only sequential access. Optical disk is more reliable than tape, generally cheaper, faster, and providing random access. Main memory is also referred to as primary storage and disks and tape as secondary storage. Stable storage represents information that has been replicated
in several nonvolatile storage media (usually disk) with independent failure modes.
For example, it may be possible to simulate stable storage using RAID (Redundant Array of Independent Disks) technology, which guarantees
that the failure of a single disk, even during data transfer, does not result in loss of data.
108
There are many different
types of failure that can affect database processing, each of which has to be dealt with in a different manner.
Some failures affect main memory only, while others involve nonvolatile (secondary) storage.
The
main

causes of failure are: • System crashes due to hardware or software errors, resulting in loss of main memory. • Media failures, such as head crashes or unreadable media, resulting in the loss of parts of secondary storage. • Application software errors, such as logical errors in the program that is accessing the database, that cause one or more transactions to fail. • Natural physical disasters, such as fires, floods, earthquakes, or power failures. • Carelessness or unintentional destruction of data or facilities by operators or users. • Sabotage, or
intentional corruption or destruction of data, hardware, or software facilities.
The loss of
main memory including database buffers, and the loss of the disk copy of the database are the
two principal effects that are to be considered regardless of the cause of the
failure. 6.3 Recovery & Transactions Transactions represent the basic unit of recovery in a database system. It is the role of the recovery manager to guarantee two of the four ACID properties of transactions, namely atomicity and durability, in the presence of failures. The recovery manager has to ensure that, on recovery from failure, either all the effects of a given transaction are permanently recorded in the database or none of them are. The situation is complicated by the fact that database writing is not an
109 atomic (single-step) action, and it is therefore possible for a transaction to have committed but for its effects not to have been permanently recorded in the database simply because they have not yet reached the database.
The database system resides permanently on nonvolatile storage (usually disks) and
only parts of the database are in memory at any time. The database
is partitioned into fixed-length storage units called blocks. Blocks are the units of data transfer to and from disk, and may contain several data items.
Transactions input information from the disk to main memory, and then output the information back onto the disk. The input and output operations are done in block units. The blocks residing on the disk are referred to as physical blocks; the blocks residing temporarily in main memory are referred to as buffer blocks. The area of memory where blocks reside temporarily is called the disk buffer.
Figure 6.1 illustrates the block storage operations.

the following

Figure 6.1 Block storage operations
The database buffers occupy an area in main memory from which data is transferred to and from secondary storage. Only once the buffers have been
110 flushed to secondary storage can any update operations be regarded as permanent. This flushing of the buffers to the database can be triggered by a specific command (for example, transaction commit) or automatically when the buffers become full. The explicit writing of the buffers to secondary storage is known as force-writing. If a failure occurs between writing to the buffers and flushing the buffers to secondary storage, the recovery manager must determine the status of the transaction that performed the write at the time of failure. If the transaction had issued its commit, then to ensure durability the recovery manager would have to redo that transaction's updates to the database (also known as rollforward). On the other hand, if the transaction had not committed at the time of failure, then the recovery manager would have to undo (rollback) any effects of that transaction on the database to guarantee transaction atomicity. If only one transaction has to be undone, this is referred to as partial undo. A partial undo can be triggered by the scheduler when a transaction is rolled back and restarted as a result of the concurrency control protocol, as described in the previous section. A transaction can also be aborted unilaterally, for example, by the user or by an exception condition in the application program. When all active transactions have to be undone, this is referred to as global undo.
The management of the database buffers plays an important role in the recovery process.

The buffer manager is responsible for the efficient management of the database buffers that are used to transfer pages to and from secondary storage. This involves reading pages from disk into the buffers until the buffers become full and then using a replacement strategy to decide which buffer(s) to force-write to disk to make space for new pages that need to be read from disk.
The
replacement strategies are first-in-first-out (FIFO) and least recently used (LRU). In addition, the buffer manager should not read a page from disk if it is already in a database buffer.
Buffer Management
111
One approach is to associate two variables with the management information for each database buffer: pinCount and dirty, which are initially set to zero for each database buffer. When a page is requested from disk, the buffer manager will check to see whether the page is already in one of the database buffers. If it is not, the buffer manager will: 1. use the replacement strategy to choose a buffer for replacement (also called the replacement buffer) and increment its pinCount. The requested page is now pinned in the database buffer and cannot be written back to disk yet. The replacement strategy will not choose a buffer that has been pinned; 2. if the dirty variable for the replacement buffer is set, it will write the buffer to disk; 3. read the page from disk into the replacement buffer and reset the buffer's dirty variable to zero. If the same page is requested again, the appropriate pinCount is incremented by 1. When the system informs the buffer manager that it has finished with the page, the appropriate pinCount is decremented by 1. At this point, the system will also inform the buffer manager if it has modified the page and the dirty variable is set accordingly. When a pinCount reaches zero, the page is unpinned and the page can be written back to disk if it has been modified (that is, if the dirty variable has been set). The following terminology is used in database recovery when pages are written back to disk: • A steal policy allows the buffer manager to write a buffer to disk before a transaction commits (the buffer is unpinned). In other words, the buffer manager "steals" a page from the transaction. The alternative policy is no- steal. • A
force policy
ensures that
all pages updated by a transaction are immediately written to disk when the transaction commits.
The alternative
112 policy is no-force. The simplest approach from an implementation perspective is to use a no-steal, force policy. With no-steal no changes of an aborted transaction
are undone
because the changes will not have been written to disk, and with force the changes of a committed transaction
are not redone
if there is a subsequent crash because all the changes will have been written to disk at commit. The deferred update recovery protocol, discussed lately, uses a no-steal policy. On the other hand, the
steal policy
avoids the need for a very large buffer space to store all updated pages
by a set of concurrent transactions, which in practice may be unrealistic anyway. In addition, the no-force policy has the distinct advantage of not having to rewrite a page to disk for a later transaction that has been updated by an earlier committed transaction and may still be in a database buffer. For these reasons, most DBMSs employ a steal, no-force policy. 6.4
Provisions for
Recovery A DBMS must provide the following provisions to assist with recovery: • A backup mechanism, which makes periodic backup copies of the database. • Logging facilities, which
keep track of the current state of transactions and database
changes. • A checkpoint facility, which enables updates to the database that are in progress to be made permanent. • A recovery manager, which allows the system to restore the database to a consistent state following a failure.
Let's discuss each one of them in detail.
Backup mechanism The DBMS should provide a mechanism to allow backup copies of the database and the log file, that should be made at regular intervals without necessarily
113 having to stop the system first. The backup copy of the database can be used in the event that the database has been damaged or destroyed. A backup can be a complete copy of the entire database or an incremental backup, consisting only of modifications made since the last complete or incremental backup. Typically, the backup is stored on offline storage, such as
an optical disk.

Log file To keep track of database transactions, the DBMS maintains a special file called a log, that contains information about all updates to the database.
Table 6.1 shows the list of data that the log may contain. Table 6.1 List of data that
the log may contain Transaction Records • Transaction identifier. • Type of log record (start, insert, update, delete, abort, commit
transaction). •
Identifier of data item affected by the database action (insert, delete, and update operations). • Before-image of the data item, that is its value before change (only update and delete operations). • After-image of the data item, that is, its value after change (only insert and update operations). • Log management information, such as a pointer to previous and next log records for that transaction (all operations). Checkpoint Records
Checkpoints are used to improve database recovery. At a checkpoint, all modified buffer blocks, all log records, and a checkpoint record identifying all active transactions are written to disk. If a failure occurs, the checkpoint record identifies which transactions need to be redone.
Owing to the importance of the transaction log file in the recovery process, the log may be duplexed or triplexed (that is, two or three separate copies are maintained) so that if one copy is damaged, another can be used. In the past, log
114 files were stored on magnetic tape because tape was more reliable and cheaper than magnetic disk. However, nowadays DBMSs are expected to be able to recover quickly from minor failures. This requires that the log file be stored online on a fast direct-access storage device.
Checkpointing The information in the log file is used to recover from a database failure. One difficulty with this scheme is that when a failure occurs
it is
not known how far back in the log to search and it may
be possible to redo
transactions that have been safely written to the database. To limit the amount of searching and subsequent processing that is needed to be carried out on the log
file,
the technique of checkpointing can be used.
Checkpoints are scheduled at predetermined intervals and involve the following operations: • Writing all log records in main memory to secondary storage. • Writing the modified blocks in the database buffers to secondary storage. • Writing a checkpoint record to the log file. This record contains the identifiers of all transactions that are active at the time of the checkpoint. If transactions are performed serially, when a failure occurs we check the log file to find the last transaction that started before the last checkpoint. Any earlier transactions would have committed previously and would have been written to the database at the checkpoint. Therefore, we need
to
redo
only the one that was active at the checkpoint and any subsequent transactions for which both start and commit records appear in the log.
If a transaction is active at the time of failure, the transaction must be
undone if
it started before the last checkpoint. If transactions
are performed concurrently, we redo all transactions that have committed since the checkpoint and undo
all transactions that were active at the time of the
crash.
115
Recovery Manager
If a failure occurs during the transaction, then the database could be inconsistent. It is the task of the recovery manager to ensure that the database is restored to the state it was in before the start of the transaction, and therefore a consistent state. 6.5

Some
of them are discussed below. 1.

93% **MATCHING BLOCK 77/202** W

Transaction failure: There are two types of errors that may cause a transaction to fail: • Logical error • : The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded. System error 2. System crash: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt.

98% **MATCHING BLOCK 78/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Transaction failure: There are two types of errors that may cause a transaction to fail: • Logical error • : The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded. System error 2. System

100% **MATCHING BLOCK 79/202** SA DBMS_AIML_FINAL.pdf (D111167082)

System crash: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of nonvolatile storage remains intact, and is not corrupted. The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption. Well-designed systems have numerous internal checks, at the hardware and the software level, that bring the system to a halt when there is an error. Hence, the fail-stop assumption is a reasonable one. :

The system has entered an undesirable state (for example, a

100% **MATCHING BLOCK 80/202** SA DBMS_AIML_FINAL.pdf (D111167082)

deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be re-executed at a later time. 116 3.

100% **MATCHING BLOCK 81/202** W

Disk failure: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or

100% **MATCHING BLOCK 83/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Disk failure: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or

on the

100% **MATCHING BLOCK 82/202** W

media, such as DVD or tapes, are used to recover from the failure.

65% **MATCHING BLOCK 84/202** SA DBMS_AIML_FINAL.pdf (D111167082)

media, such as DVD or tapes, are used to recover from the failure. There is a need to identify the failure modes of

the respective devices used for storing data. This helps in determining how the system should recover from failures. Next step to be considered is

75%	MATCHING BLOCK 85/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

how these failure modes affect the contents of the database. Accordingly, algorithms can be proposed to ensure database consistency and transaction atomicity despite failures. These algorithms

are known as recovery algorithms and

100%	MATCHING BLOCK 86/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

have two parts: 1. Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures. 2. Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability. 6.6 Recovery

Techniques The procedure for recovery depends
on the extent of the damage that has occurred to the database. Let's consider two cases below: 1. If the database has been extensively damaged:
Consider an
example, if
a disk head has crashed and destroyed the database, then it is necessary to restore the last backup copy of the database on a replacement disk
and reapply the update operations of committed transactions using the log file. This assumes that the log file has not been damaged as well.
It is
recommended that where possible the log file be stored on a disk separate from the main database files. This reduces the risk of both the database files and the log file being damaged at the same time. 2. If the database has not been physically damaged but has become inconsistent:
117
Consider an
example, if
the system crashed while transactions were executing, then it is necessary to undo the
changes that caused the inconsistency. It may also be necessary to redo some
transactions to ensure that the updates they performed have reached secondary storage. Here,
there is no
need to use the backup copy of the database but can restore the database to a consistent state using the before- and after-images held in the log
file.
The second case is considered to look at the two techniques for recovery.
The techniques, known as deferred update and immediate update, differ in the way that updates are written to secondary storage. An alternative technique called shadow paging
is also discussed briefly.
Recovery techniques using deferred update Using the deferred update recovery protocol, updates are not written to the database until after a transaction has reached its commit point.
If a transaction fails before it reaches this point, it will not have modified the database
and so no undoing of changes will be necessary. However,
it may be necessary to redo the updates of committed transactions as their effect may not have reached the database.
In this case, the log file is used to protect against system failures in the following ways: • When a transaction starts, write a transaction start record to the log. • When any write operation is performed, write a log record containing all the log data specified previously (excluding the before-image of the update). Do not actually write the update to the database buffers or the database itself. • When a transaction is about to commit, write a transaction commit log record, write all the log records for the transaction to disk, and then commit the transaction. Use the log records to perform the actual updates to the database. • If a transaction aborts, ignore the log records for the transaction and do not perform the writes.
118 It should be noted that the log records
are written

to disk before the transaction is actually committed, so that if a system failure occurs while the actual database updates are in progress, the log records will survive and the updates can be applied later. In the event of a failure, it is noted that the log to identify the transactions that were in progress at the time of failure. Starting at the last entry in the log file, the most recent checkpoint record
is approached.
Any transaction with transaction start and transaction commit log records should be redone. The redo procedure performs all the writes to the database using the after image log records for
the transactions, in
the order in which they were written to the-log.
If this writing has been performed already, before the failure, the write has no effect on the data item, so there is no damage done if the data
is written
again (that is, the operation is idempotent). However, this method guarantees that we will update any data item that was not properly updated prior to the failure. For any transactions with transaction start and transaction abort log records, nothing is done,
because
no actual writing was done to the database, so these transactions do not have to be undone. If a second system crash occurs during recovery, the log records are used again to restore the database. With the form of the write log records, it does not matter how many times the writes
are redone.
Recovery techniques using immediate update Using the immediate update recovery protocol, updates are applied to the database as they occur without waiting to reach the commit point. As well as having to redo the updates of committed transactions following a failure, it may now be necessary to undo the effects of transactions that had not committed at the time of failure. In this case, the log file is used to protect against system failures in the following way: • When a transaction starts, write a transaction start record to the log.
119 • When a write operation is performed, write a record containing the necessary data to the log file. • Once the log record is written, write the update to the database buffers. • The updates to the database itself are written when the buffers are next flushed to secondary storage. • When the transaction commits, write a transaction commit record to the log. It is essential that log records are written before the corresponding write to the database. This is known as the write-ahead log protocol. If updates were made to the database first and failure occurred before the log record was written, then the recovery manager would have no way of undoing the operation. Under the write-ahead log protocol, the recovery manager can safely assume that if there is no transaction commit record in the log file for a particular transaction, then that transaction was still active at the time of failure and must therefore be undone. If a transaction aborts, the log can be used to undo it, as it contains all the old values for the updated fields. As a transaction may have performed several changes to an item, the writes are undone in reverse order. Regardless of whether the transaction's writes have been applied to the database itself, writing the before-images guarantees
that
the database is
restored to its state prior to the start of the transaction.
If the system fails, recovery involves using the log to undo or redo transactions. For any transaction for which both a transaction start and transaction commit record appear in the log, we redo using the log records to write the after-image of updated fields.
For any transaction for which
the log contains a transaction start record but not a
transaction commit record,
that transaction is undone.
The log records are used to write the before-image of the affected fields, and thus restore the database to its state prior to the transaction's start. The undo operations are performed in the reverse order to which they were written to the log.
120 Shadow paging An alternative to the log-based recovery schemes described above is shadow paging,
introduced by Lorie, 1977. In this scheme, two-page tables are maintained
during the life of a transaction: a current page table and a shadow page table.
When the transaction starts, the two-page tables are the same. The shadow page table is never changed thereafter, and is used to restore the database in the event of a system failure. During the transaction, the current page table is used to record all updates to the database. When the transaction completes, the current page table becomes the shadow page table.

97% **MATCHING BLOCK 87/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Shadow paging considers the database to be made up of n number of fixed-size disk pages (or disk blocks) for recovery purposes. A directory with n entries is constructed, where the i th entry points to the i th database page on disk. The directory is kept in main memory if it is not too large, and all references, reads or writes, to database pages on disk go through it. When a transaction begins executing, the current directory, whose entries point to the most recent or current database pages on disk, is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory. To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database 121 thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery. In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits.

97% **MATCHING BLOCK 88/202** SA 47F417BA15858476660.pdf (D123781188)

Shadow paging considers the database to be made up of n number of fixed-size disk pages (or disk blocks) for recovery purposes. A directory with n entries is constructed, where the i th entry points to the i th database page on disk. The directory is kept in main memory if it is not too large, and all references, reads or writes, to database pages on disk go through it. When a transaction begins executing, the current directory, whose entries point to the most recent or current database pages on disk, is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory. To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database 121 thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery. In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits. The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation. 6.7

100% MATCHING BLOCK 99/202 SA DBMS_AIML_FINAL.pdf (D111167082)

The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation. 6.7

Recovery in Multidatabase Systems So far, we have implicitly assumed that a transaction accesses a single database. In some cases a single transaction, called a multidatabase transaction, may require access to multiple databases. These databases may even be stored on different types of DBMSs; for example, some DBMSs may be relational, whereas others are object-oriented, hierarchical, or network DBMSs. In such a case, each DBMS involved in the multidatabase transaction may have its own recovery technique and transaction manager separate from those of the other DBMSs. This situation is somewhat similar to the case of a distributed database management system, where parts of the database reside at different sites that are connected by a communication network.
122 To maintain the atomicity of a multidatabase transaction, it is necessary to have a two-level recovery mechanism. A global recovery manager, or coordinator, is needed to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables). The coordinator usually follows a protocol called the two-phase commit protocol, whose two phases can be stated as follows: • Phase 1: When all participating databases signal to the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a "ready to commit" or "OK" signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a "cannot commit" or "not OK" signal to the coordinator. If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a "not OK" response. • Phase 2: If all participating databases reply "OK," and the coordinator's vote is also "OK," the transaction is successful, and the coordinator sends a "commit" signal for the transaction to the participating databases. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, recovery from failure is now possible. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. On the other hand, if one or more of the participating databases or the coordinator have a "not OK" response, the transaction has failed, and the coordinator sends a message to "roll back" or UNDO the local effect of the transaction to each participating database. This is done by undoing the transaction operations, using the log.
123 The net effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants or the coordinator fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back. A failure during or before Phase 1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit. 6.8 Remote Backup Systems Traditional transaction-processing systems are centralized or client–server systems. Such systems are vulnerable to environmental disasters such as fire, flooding, or earthquakes. Increasingly, there is a need for transaction-processing systems that can function in spite of system failures or environmental disasters. Such systems must provide high availability; that is, the time for which the system is unusable must be extremely small. High availability can be achieved by performing transaction processing at one site, called the primary site, and having a remote backup site where all the data from the primary site are replicated. The remote backup site is sometimes also called the secondary site. The remote site must be kept synchronized with the primary site, as updates are performed at the primary. Synchronization is achieved by sending all log records from the primary site to the remote backup site. The remote backup site must be physically separated from the primary—for example, we can locate it in a different state, so that a disaster at the primary does not damage the remote backup site. Figure 6.2 shows the architecture of a remote backup system.
124 Figure 6.2 Architecture of

76% MATCHING BLOCK 89/202 W

remote backup system When the primary site fails, the remote backup site takes over processing.

First, however, it performs recovery, using its copy of the data from the primary, and the log records received from the primary. In effect, the remote backup site is performing recovery actions that would have been performed at the primary site when the latter recovered. Standard recovery algorithms, with minor modifications, can be used for recovery at the remote backup site. Once recovery has been performed, the remote backup site starts processing transactions. Availability is greatly increased over a single-site system, since the system can recover even if all data at the primary site is lost.

100% **MATCHING BLOCK 90/202** W

Several issues must be addressed in designing a remote backup system: • Detection of failure: It is important for the remote backup system to detect when the primary has failed.

Failure of communication lines can fool the remote backup into believing that the primary has failed. To avoid this problem, several communication links are maintained

100% **MATCHING BLOCK 91/202** W

with independent modes of failure between the primary and the remote backup.

For example, several independent network connections, including perhaps a modem connection over a telephone line, may be used. These connections may be backed up via manual intervention by operators, who can communicate over the telephone system. •

100% **MATCHING BLOCK 92/202** W

Transfer of control: When the primary fails, the backup site takes over processing and becomes the new primary. When the original primary site recovers, it can either play the role of remote backup, or take over the role of primary site again.

In either case, the old primary must receive a log of updates carried out by the backup site while the old primary was down. The simplest way of transferring control is for the old primary to receive redo logs from the old backup site, and to catch up with the updates by applying them locally. The old primary can then act as a remote backup site. If control 125 must be transferred back, the old backup site can pretend to have failed, resulting in the old primary taking over. •

100% **MATCHING BLOCK 93/202** W

Time to recover: If the log at the remote backup grows large, recovery will take a long time. The remote backup site can periodically process the redo log records that it has received and can perform a checkpoint, so that earlier parts of the log can be deleted. The delay before the remote backup takes over can be significantly reduced as a result. • A hot-spare configuration can make

a takeover by the backup site almost instantaneous. In this configuration, the remote backup site continually processes redo log records as they arrive, applying the updates locally. As soon as the failure of the primary is detected, the backup site completes recovery by rolling back incomplete transactions; it is then ready to process new transactions. •

100% **MATCHING BLOCK 94/202** W

Time to commit: To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site. This delay can result in a longer wait to commit a transaction, and some systems therefore permit lower degrees of durability. The degrees of durability can be classified as follows: ? One-safe ? : A transaction commits as soon as its commit log record is written to stable storage at the primary site.

The problem with this scheme is that the updates of a committed transaction may not have made it to the backup site, when the backup site takes over processing. Thus, the updates may appear to be lost. When the primary site recovers, the lost updates cannot be merged in directly, since the updates may conflict with later updates performed at the backup site. Thus, human intervention may be required to bring the database to a consistent state.

85% **MATCHING BLOCK 95/202** **W**

Two-very-safe: A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site. The problem with this scheme is

that transaction processing cannot proceed if either the
126 primary or the backup site is down. Thus, availability is actually less than in the single-site case, although the probability of data loss is much less. ? Two-safe Several commercial shared-disk systems provide a level of fault tolerance that is intermediate between centralized and remote backup systems. In these commercial systems, the failure of a CPU does not result in system failure. Instead, other CPUs take over, and they carry out recovery. Recovery actions include rollback of transactions running on the failed CPU , and recovery of locks held by those transactions. Since data is on a shared disk, there is no need for transfer of log records. :

100% **MATCHING BLOCK 96/202** **W**

This scheme is the same as two-very-safe if both primary and backup sites are active. If only the primary is active, the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site. This scheme provides better availability than does two-very-safe, while avoiding the problem of lost transactions faced by the one-safe scheme.

It results in a slower commit than the one-safe scheme, but the benefits generally outweigh the cost.

100% **MATCHING BLOCK 97/202** **W**

An alternative way of achieving high availability is to use a distributed database, with data replicated at more than one site. Transactions are then required to update all replicas of any data item that they update. 6.9

Summary •

87% **MATCHING BLOCK 98/202** **W**

An integral part of a database system is a recovery system that

94% **MATCHING BLOCK 102/202** **SA** DBMS_AIML_FINAL.pdf (D111167082)

An integral part of a database system is a recovery system that can restore the database to the consistent state that existed before the failure. •

There are different causes for a computer system to fail or crash.

100% **MATCHING BLOCK 100/202** **W**

In addition to system failures, transactions may also fail for various reasons, such as violation of integrity constraints or deadlocks. •

97% **MATCHING BLOCK 101/202** W

The various types of storage in a computer are volatile storage, nonvolatile storage, and stable storage. Data in volatile storage, such as in RAM , are lost 127 when the computer crashes. Data in nonvolatile storage, such as disk, are not lost when the computer crashes, but may occasionally be lost because of failures such as disk crashes. Data in stable storage is never lost. •

100% **MATCHING BLOCK 104/202** W

In case of failure, the state of the database system may no longer be consistent; that is, it may not reflect a state of the world that the database is supposed to capture. To preserve consistency, we require that each transaction be atomic. It is the responsibility of the recovery scheme to ensure the atomicity and durability property. •

95% **MATCHING BLOCK 103/202** SA 47F417BA15858476660.pdf (D123781188)

a state of the world that the database is supposed to capture.

95% **MATCHING BLOCK 107/202** SA DBMS_AIML_FINAL.pdf (D111167082)

a state of the world that the database is supposed to capture.

A DBMS must provide the following provisions to assist with recovery: a backup mechanism,
Logging facilities, a checkpoint facility, and a recovery manager. •

100% **MATCHING BLOCK 105/202** W

In log-based schemes, all updates are recorded on a log, which must be kept in stable storage. A transaction is considered to have committed when its last log record, which is the commit log record for the transaction, has been output to stable storage. •

Using the deferred update recovery protocol, updates are not written to the database until after a transaction has reached its commit point.
Using the immediate update recovery protocol, updates are applied to the database as they occur without waiting to reach the commit point. •

92% **MATCHING BLOCK 106/202** W

Remote backup systems provide a high degree of availability, allowing transaction processing to continue even if the primary site is destroyed. Data and log records from a primary site are continually backed up to a remote backup site.
6.10

Key Terms • Blocks: The database
is partitioned into fixed-length storage units called blocks. Blocks are the units of data transfer to and from disk, and may contain several data items.
128 •
Checkpointing:
To limit the amount of searching and subsequent processing that is needed to be carried out on the log
file,
the technique of checkpointing can be used. • Fail-stop Assumption:

100% **MATCHING BLOCK 112/202** SA DBMS_AIML_FINAL.pdf (D111167082)

The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption. •

Deferred Update Recovery Protocol:
Using the deferred update recovery protocol, updates are not written to the database until after a transaction has reached its commit point. •
Recovery Update Recovery Protocol:
Using the immediate update recovery protocol, updates are applied to the database as they occur without waiting to reach the commit point. 6.11
Check Your Progress Short- Answer Type Q1) Log records include: a) New value b) Old value c) Both A & B d) None of the above Q2) _____ blocks reside on the disk. Q3) Recovery scheme helps in restoring the database to the consistent state before failure. (True/ False?) Q4) _____ causes loss of data of volatile storage, and halts the transaction processing. Q5) The types of blocks that reside temporarily in the main memory are: a) Physical blocks b) Buffer blocks c) Logical blocks d) None of the above
129 Long- Answer Type Q1) Explain the shadow paging recovery technique. Q2) What are the essential facilities available for recovery in a DBMS? Q3) Describe the various types of Recovery techniques. Q4)
Explain the difference between the three storage types—volatile, non-volatile, and stable
storage. Q5) Describe the two-phase commit protocol for multi database transactions. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011. •
Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007.
130 MODULE- IV CONCURRENCY CONTROL &
LOCKING SYSTEMS
131
Unit: 07 Concurrency Control Structure 7.0 Introduction 7.1 Unit Objectives 7.2 Need for Concurrency Control 7.3 Concurrency Control by Timestamps 7.4 Concurrency Control by Validation 7.5 Multiversion Schemes 7.6 Snapshot Isolation 7.7 Summary 7.8 Key Terms 7.9 Check Your Progress 7.0 Introduction It is already known that one of the fundamental properties of a transaction is isolation.

68% **MATCHING BLOCK 108/202** W

When many transactions execute concurrently in the database, the isolation property may no longer be preserved.

To ensure that it is, the interaction among the concurrent transactions should be controlled by the system. Concurrency control schemes are one of the most used mechanisms
to control the interactions. It
is the
process of managing simultaneous operations on the database without having them interfere with one another.
Most of these techniques ensure serializability of schedules, using certain protocols. One important set of protocols define locking data items to prevent multiple transactions from accessing the items concurrently. Locking protocols are used in most commercial DBMSs. Locking system is discussed in the next unit in detail. Another set of concurrency control protocols use timestamps. A
132 timestamp is a unique identifier for each transaction, generated by the system. Other protocols are based on the concept of validation or certification of a transaction after it executes its operations; these are sometimes called optimistic protocols. This unit focuses on the need and different concurrency control techniques. It also depicts the concurrency control issues that arise when indexes are used to process transactions. 7.1 Unit Objectives After completing this unit, the reader will be able to: • Gain knowledge about the need for concurrency control in databases. • Discuss concurrency control by timestamps and validation. • Learn about the concurrent control in multiversion schemes. 7.2 Need for Concurrency Control A major objective in developing a database is to enable many users to access shared data concurrently. Concurrent access is relatively easy if all users are only reading data, as there is no way that they can interfere with one another. However, when two or more users are accessing

the database simultaneously and at least one is updating data, there may be interference that can result in inconsistencies. This objective is similar to the objective of multi-user computer systems, which allow two or more programs (or transactions) to execute at the same time. For example, many systems have input/output (I/O) subsystems that can handle I/O operations independently, while the main central processing unit (CPU) performs other operations. Such systems can allow two or more transactions to execute simultaneously.
133
Concurrency control in DBMS
ensures that database transactions are performed concurrently without violating the data integrity of the respective databases. Thus concurrency control is an essential element for correctness in any system where two database transactions or more, executed with time overlap, can access the same data,
e.g., virtually in any general-purpose database system. A well-established concurrency control theory exists for database systems: serializability theory, which allows effectively designing and analyzing concurrency control methods and mechanisms. To ensure correctness, a DBMS usually guarantees that only serializable transaction schedules are generated, unless serializability is intentionally relaxed. For maintaining correctness in cases of failed (aborted) transactions (which can always happen for many reasons) schedules also need to have the recoverability property. A DBMS also guarantees that no effect of committed transactions is lost, and no effect of aborted (rolled back) transactions remains in the related database. Concurrency control is a database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. Concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity. The concurrency is about controlling the multi-user access of the database. There are three basic object concurrency control strategies: pessimistic, optimistic and truly optimistic. • Pessimistic concurrency control locks the source for the entire time that a copy of it exists, not allowing other copies to exist until the copy with the lock has finished its transaction. The copy effectively places a write lock on the appropriate source, performs some work, then applies the appropriate changes to the source and unlocks it. This is a brute force approach to concurrency that is applicable for small-scale systems or systems where concurrent access
134 is very rare: Pessimistic locking doesn't scale well because it blocks simultaneous access to common resources. • Optimistic concurrency control takes a different approach, one that is more complex but is scalable. With optimistic control, the source is uniquely marked each time it's initially accessed by any given copy. The access is typically a creation of the copy or a refresh of it. The user of the copy then manipulates it as necessary. When the changes are applied, the copy briefly locks the source, validates that the mark it placed on the source has not been updated by another copy, commits its changes and then unlocks it. When a copy discovers that the mark on the source has been updated, indicating that another copy of the source has since accessed it, we say that a collision has occurred. A similar but alternative strategy is to check the mark placed on the object previously, if any, to see that it is unchanged at the point of update. The value of the mark is then updated as part of the overall update of the source. The software is responsible for handling the collision appropriately. Since it's unlikely that separate users will access the same object simultaneously, it's better to handle the occasional collision than to limit the size of the system. This approach is suitable for large systems or for systems with significant concurrent access. • Truly optimistic concurrency control is the most simplistic and effectively unusable for most systems. With this approach, no locks are applied to the source and no unique marks are placed on it; the software simply hopes for the best and applies the changes to the source as they occur. This approach means your system doesn't guarantee transactional integrity if it's possible that two or more users can manipulate a single object simultaneously. Truly optimistic concurrency control is only appropriate for systems that have no concurrent update at all, such as information-only Web sites or single user systems.
135 7.3 Concurrency Control by Timestamps
Locking methods are the most widely used approach to ensure serializability of concurrent transactions.
The lock prevents another transaction from modifying the item or even reading it, in the case of an exclusive lock.
The
order of transactions in the equivalent serial schedule is based on the order in which the transactions lock the items they require. If a transaction needs an item that is already locked, it may be forced to wait until the item is released. A different approach that also guarantees serializability uses transaction timestamps to order
transaction
execution for an equivalent serial schedule. Timestamp methods for concurrency control are quite different from locking methods. No locks are involved and therefore there can be no deadlock. Locking methods generally prevent conflicts by making transactions wait. With timestamp methods, there is no waiting:
transactions involved in conflict are simply rolled back and restarted. A
timestamp is
a unique identifier created by the DBMS that indicates the relative starting time of a transaction.

Timestamps can be generated by simply using the system clock at the time the transaction started, or, more normally, by incrementing a logical counter every time a new transaction starts.
Timestamping
is
a concurrency control protocol that orders transactions in such a way that older transactions, transactions with smaller timestamps,
get priority in the event of conflict. With timestamping, if a transaction attempts to read or write a data item, then
the read or write is only allowed to proceed
if the last update on that data item was carried out by an older transaction. Otherwise, the transaction requesting the read/ write is restarted and given a new timestamp.
New timestamps must be assigned to restarted transactions to prevent their being continually aborted and restarted. Without new timestamps, a transaction with an old timestamp might not be able
to commit owing to younger transactions having already committed.
136
In addition to
timestamps for transactions, there are timestamps for data items. Each data item contains a read_timestamp, giving the timestamp of the last transaction to read the item, and a write_timestamp, giving the timestamp of the last transaction to write (update) the item. •

90% **MATCHING BLOCK 109/202** W

W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. • R-timestamp(Q) denotes the largest timestamp of any transaction that executed read(Q) successfully. These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed. The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows: 1. Suppose that transaction T i issues read(Q). a) If TS(T i) > W-timestamp(Q), then T i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T i is rolled back. b) If TS(T i) ≥ W-timestamp(Q), then the read operation is executed, and R- timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T i). 2. Suppose that transaction T i issues write(Q). a) If TS(T i) > R-timestamp(Q), then the value of Q that T i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls T i back. b) If TS(T i) > W-timestamp(Q), then T i is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls T i back. c) Otherwise, the system executes

the

93% **MATCHING BLOCK 110/202** W

write operation and sets W- timestamp(Q) to TS(T i). 137 If a transaction T i is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

This
scheme, called basic timestamp ordering, guarantees that transactions are conflict serializable, and the results are equivalent to a serial schedule in which the transactions are executed in chronological order
of the timestamps. In other words, the results will be as if all of transaction 1 were executed, then all of transaction 2, and so on, with no
interleaving.

100% **MATCHING BLOCK 111/202** W

The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order.

The protocol ensures freedom from deadlock, since no transaction ever waits.

77% **MATCHING BLOCK 115/202** SA DBMS_AIML_FINAL.pdf (D111167082)

However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction.

If a transaction is suffering from repeated restarts, conflicting transactions need to be temporarily blocked to enable the transaction to finish.
Thomas's write rule
A modification to the basic timestamp ordering protocol that relaxes conflict serializability can be used to provide greater concurrency by rejecting obsolete write operations (
Thomas, 1979). The
extension, known as Thomas's write rule, modifies the checks for a write operation by transaction T i as follows:

88% **MATCHING BLOCK 113/202** W

Suppose that transaction T i issues write(Q). 1. If TS(T i) > R-timestamp(Q), then the value of Q that T i is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls T i back. 2. If TS(T i) > W-timestamp(Q), then T i is attempting to write an obsolete value of Q. Hence, this write operation can be ignored. 3. Otherwise, the system executes the write operation and

sets W-timestamp(Q) to TS(T
i).
138 7.4 Concurrency Control by Validation

91% **MATCHING BLOCK 114/202** W

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions

95% **MATCHING BLOCK 117/202** SA DBMS_AIML_FINAL.pdf (D111167082)

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low. Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state. A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead. A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for monitoring the system. The validation protocol requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order: 1. Read phase: During this phase, the system executes transaction T i . It reads the values of the various data items and stores them in variables local to T i . It performs all write operations on temporary local variables, without updates of the actual database. 2. Validation phase: The validation test is applied to transaction T i . This determines whether T i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction. 3. Write phase: If the validation test succeeds for transaction T i , the temporary local variables that hold the results of any write operations performed by T i are copied to the database. Read-only transactions omit this phase.

92% **MATCHING BLOCK 116/202** W

The validation protocol requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order: 1. Read phase: During this phase, the system executes transaction T i . It reads the values of the various data items and stores them in variables local to T i . It performs all write operations on temporary local variables, without updates of the actual database. 2. Validation phase: The validation test is applied to transaction T i . This determines whether T i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction. 3. Write phase: If the validation test succeeds for transaction T i , the temporary local variables that hold the results of any write operations performed by T i are copied to the database. Read-only transactions omit this phase.

Validation Test

89% **MATCHING BLOCK 118/202** W

Each transaction must go through the phases in the order shown. However, phases of concurrently executing transactions can be interleaved. To perform the validation test, we need to know when the various phases of transactions took 139 place. We shall, therefore, associate three different timestamps with each transaction T i : • Start(T i), the time when T i started its execution. • Validation(

92% **MATCHING BLOCK 119/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Each transaction must go through the phases in the order shown. However, phases of concurrently executing transactions can be interleaved. To perform the validation test, we need to know when the various phases of transactions took 139 place. We shall, therefore, associate three different timestamps with each transaction T i : • Start(T i), the time when T i started its execution. • Validation(

T i),

89% **MATCHING BLOCK 120/202** W

the time when T i finished its read phase and started its validation phase. • Finish(

89% **MATCHING BLOCK 121/202** SA DBMS_AIML_FINAL.pdf (D111167082)

the time when T i finished its read phase and started its validation phase. • Finish(

T i),

75% **MATCHING BLOCK 123/202** SA DBMS_AIML_FINAL.pdf (D111167082)

the time when T i finished its write phase. We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation(T i). Thus, the value TS(T i) = Validation(T i) and, if TS(

T j) > TS(T k),

61% **MATCHING BLOCK 124/202** SA DBMS_AIML_FINAL.pdf (D111167082)

then any produced schedule must be equivalent to a serial schedule in which transaction T j appears before transaction T k . The reason we have chosen Validation(T i), rather than Start(

76% **MATCHING BLOCK 122/202** W

equivalent to a serial schedule in which transaction T j appears before transaction

T i),

91% **MATCHING BLOCK 126/202** SA DBMS_AIML_FINAL.pdf (D111167082)

as the timestamp of transaction T i is that we can expect faster response time provided that conflict rates among transactions are indeed low. The validation test for transaction T i requires that, for all transactions

62% **MATCHING BLOCK 125/202** W

The validation test for transaction T i requires that, for all transactions T k with TS(T k) > TS(T i), one of the following two conditions must hold: 1. Finish(

T k) > Start(T i). Since T k completes its execution before T i starts,

64% **MATCHING BLOCK 127/202** W

the serializability order is indeed maintained. 2. The set of data items written by T k does not

64% **MATCHING BLOCK 128/202** SA DBMS_AIML_FINAL.pdf (D111167082)

the serializability order is indeed maintained. 2. The set of data items written by T k does not

intersect with the set of data items read by T i , and T k completes its write phase before T i starts its validation phase (Start(T i) > Finish(T k) >

43% **MATCHING BLOCK 129/202** W

Validation(T i)). This condition ensures that the writes of T k and T i do not overlap. Since the writes of T k do not affect the read of T i , and since T i cannot affect the read of T k , the serializability order is indeed maintained. The validation scheme

43% **MATCHING BLOCK 131/202** SA DBMS_AIML_FINAL.pdf (D111167082)

Validation(T i)). This condition ensures that the writes of T k and T i do not overlap. Since the writes of T k do not affect the read of T i , and since T i cannot affect the read of T k , the serializability order is indeed maintained.

97% **MATCHING BLOCK 138/202** SA DBMS_AIML_FINAL.pdf (D111167082)

The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed. However, there is a possibility of starvation of long transactions, due to a sequence of conflicting short transactions that cause repeated restarts of the 140 long transaction. To avoid starvation, conflicting transactions must be temporarily blocked, to enable the long transaction to finish. This validation scheme is called the optimistic concurrency-control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end. In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conflict is detected, even though there is a chance that the schedule may be conflict serializable. 7.5

100% MATCHING BLOCK 130/202 W

This validation scheme is called the optimistic concurrency-control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end.

Multiversion Schemes The concurrency-control schemes discussed so far ensure serializability by either delaying an operation or aborting the transaction that issued the operation. For example, a read operation may be delayed because the appropriate value has not been written yet; or it may be rejected (that is, the issuing transaction must be aborted) because the value that it was supposed to read has already been overwritten.
These difficulties could be avoided if old copies of each data item were kept in a system. In multiversion concurrency-control schemes, each write(Q) operation creates a new version of Q. When a transaction issues a read(Q) operation, the concurrency-control manager selects one of the versions of Q

87% MATCHING BLOCK 132/202 W

to be read. The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability.

It is also crucial, for performance reasons, that a transaction be able to determine easily and quickly which version of the data item should be read. The timestamp-ordering protocol can be extended to a multiversion protocol.

81% MATCHING BLOCK 133/202 W

With each transaction T i in the system, we associate a unique static timestamp, denoted by TS(

T i). The database system assigns this timestamp before the transaction starts execution. With each data item Q, a sequence of versions >Q 1 , Q 2 , . . . , Q m < is associated.
141 Each version Q k

78% MATCHING BLOCK 134/202 W

contains three data fields: • Content is the value of version Q k . • W-timestamp(

Q k) is the timestamp of the transaction that created version Q k . • R-timestamp(Q k)

100% MATCHING BLOCK 135/202 W

is the largest timestamp of any transaction that successfully read version

Q k . A transaction T i creates a new version Q k of data item Q by issuing a write(Q) operation. The content field of the version holds the value written by T i . The system initializes the W-timestamp and R-timestamp to TS(T i). It updates the R- timestamp value of Q k whenever a transaction T j reads the content of Q k , and R- timestamp(Q k) > TS(T j). The multiversion timestamp-ordering scheme presented next ensures serializability. The scheme operates as follows: Suppose that transaction T i issues a read(Q) or write(Q) operation. Let Q k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS(T i). 1. If transaction T i issues a read(Q), then the value returned is the content of version Q k . 2. If transaction T i issues write(Q), and if TS(T i) > R-timestamp(Q k), then the system rolls back transaction T i . On the other hand, if TS(T i) = W- timestamp(Q k), the system overwrites the contents of Q k ; otherwise (if TS(T i) < R-timestamp(Q k)), it creates a new version of Q. The justification for rule 1 is clear. A transaction reads the most recent version that comes before it in time. The second rule forces a transaction to abort if it is “too late” in doing a write. More precisely, if T i attempts to write a version that some other transaction would have read, then we cannot allow that write to succeed. Versions that are no longer needed are removed according to the following rule: Suppose that there are two versions, Q k and Q j , of a data item, and that both
142 versions have a W-timestamp less than
the timestamp of the oldest transaction in the system. Then,

the older of the two versions Q k and Q j will not be used again, and can be deleted. The multiversion timestamp-ordering scheme has the desirable property that a read request never fails and is never made to wait. In typical database systems, where reading is a more frequent operation than is writing, this advantage may be of major practical significance. 7.6 Snapshot Isolation Snapshot isolation is a particular type of concurrency-control scheme that has gained wide acceptance in commercial and open-source systems, including Oracle, PostgreSQL, and SQL Server. Conceptually, snapshot isolation involves giving a transaction a "snapshot" of the database at the time when it begins its execution. It then operates on that snapshot in complete isolation from concurrent transactions. The data values in the snapshot consist only of values written by committed transactions. This isolation is ideal for read-only transactions since they never wait and are never aborted by the concurrency manager. Transactions that update the database must, of course, interact with potentially conflicting concurrent update transactions before updates are actually placed in the database. Updates are kept in the transaction's private workspace until the transaction successfully commits, at which point the updates are written to the database. When a transaction T is allowed to commit, the transition of T to the committed state and the writing of all of the updates made by T to the database must be done as an atomic action so that any snapshot created for another transaction either includes all updates by transaction T or none of them.
143 7.7
Summary •
Concurrency control is the
process of managing simultaneous operations on the database without having them interfere with one another. •

100% **MATCHING BLOCK 136/202** W

To ensure serializability, we can use various concurrency-control schemes. All these schemes either delay an operation or abort the transaction that issued the operation. The most common ones are locking protocols, timestamp ordering schemes, validation techniques, and multiversion schemes. •

76% **MATCHING BLOCK 137/202** W

A timestamp-ordering scheme ensures serializability by selecting an ordering in advance between every pair of transactions. The timestamps of the transactions determine the serializability order. •

100% **MATCHING BLOCK 139/202** W

A validation scheme is an appropriate concurrency-control method in cases where a majority of transactions are read-only transactions, and thus the rate of conflicts among these transactions is low.

100% **MATCHING BLOCK 144/202** SA DBMS_AIML_FINAL.pdf (D111167082)

in cases where a majority of transactions are read-only transactions,

100% **MATCHING BLOCK 140/202** W

The serializability order is determined by the timestamp of the transaction. A transaction in this scheme is never delayed. •

100% **MATCHING BLOCK 141/202** W

A multiversion concurrency-control scheme is based on the creation of a new version of a data item for each transaction that writes that item. When a read operation is issued, the system selects one of the versions to be read. •

100% MATCHING BLOCK 142/202 W

Snapshot isolation is a multiversion concurrency-control protocol based on validation, which, unlike multiversion two-phase locking, does not require transactions to be declared as read-only or update. Snapshot isolation does not guarantee serializability, but is nevertheless supported by many database systems. 7.8

Key Terms • Timestamp: A timestamp is a unique identifier for each transaction, generated by the system.
144 • Pessimistic concurrency control: It locks the source for the entire time that a copy of it exists, not allowing other copies to exist until the copy with the lock has finished its transaction. •

90% MATCHING BLOCK 143/202 W

W-timestamp(Q): It denotes the largest timestamp of any transaction that executed write(Q) successfully. • R-timestamp(Q): It denotes the largest timestamp of any transaction that executed read(Q) successfully. •

Validation Protocol: It

94% MATCHING BLOCK 145/202 W

requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. 7.9

94% MATCHING BLOCK 159/202 SA DBMS_AIML_FINAL.pdf (D111167082)

requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. 7.9

Check Your Progress Short- Answer Type Q1) A concurrency control scheme that creates a new version of data item for each transaction: a) Multiversion b) Validation c) Timestamp d) None of the above Q2) _____ is a challenging task for transactions that have user interactions. Q3) Snapshot isolation is a scheme of Multiversion concurrency control. (True/ False?) Q4) A timestamp ordering scheme ensures: a) Atomicity b) Cascading c) Serializability d) None of the above Q5) Snapshot isolation scheme supports serializability. (True/ False?) Long- Answer Type Q1) Discuss why concurrency control is needed. Support with examples.
145 Q2) Discuss the difference between pessimistic and optimistic concurrency control. Q3)
Describe the basic timestamp ordering protocol for concurrency control. What is Thomas's write rule and how does this affect the basic timestamp ordering protocol?
Q4) Write a short note on Multiversion schemes of concurrency control. Q5) Define Validation Protocol. Explain the validation test for concurrency control. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
146 Unit: 08 Locking Systems Structure 8.0 Introduction 8.1 Unit Objectives 8.2 Locking Protocols 8.2.1 Two-Phase Locking (2PL) 8.3 Implementation of Locking 8.4 Granularity of Data Items 8.4.1 Multiple Granularity Locking 8.5 Concurrency control in index structures using locks 8.6 Other Concurrency Control Issues 8.6.1 Insertion, Deletion, and Phantom Records 8.6.2 Interactive Transactions 8.6.3 Latches 8.7 Summary 8.8 Key Terms 8.9 Check Your Progress 8.0 Introduction We are already aware that
concurrency control is the
process of managing simultaneous operations on the database without having them interfere with one another.
Also, we have studied a variety of concurrency-control schemes like timestamping, validation scheme, multiversion schemes, and snapshot isolation scheme. Locking scheme is the most widely used method of
concurrency control. One of the main techniques used to control concurrent execution of transactions
147 (that is to provide serializable execution of transactions) is based on
the concept of locking data items.
A lock is a variable associated with a data item

in the database and describes the status of that data item
with respect to possible operations that can be applied to the item. Generally speaking, there is one lock for each data item in the database. The overall purpose of locking is to obtain maximum concurrency and minimum delay in processing transactions.

64% **MATCHING BLOCK 146/202** W

When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.

100% **MATCHING BLOCK 147/202** W

One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. 8.1

Unit Objectives After completing this unit, the reader will be able to: ● Acquire knowledge of locking systems and their need. ● Learn about the different locking
protocols. ● Discuss
the Granularity of data items. 8.2 Locking Protocols A
lock is a variable associated with a data item that
describes the status of the item
with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database.
Locks are used as a
means of synchronizing the access by concurrent transactions to the database items.
There are two modes in which a data item may be locked. 1.

91% **MATCHING BLOCK 148/202** W

Shared: If a transaction T i has obtained a shared-mode lock (denoted by S) 148 on item Q, then

T i

93% **MATCHING BLOCK 149/202** W

can read, but cannot write, Q. 2. Exclusive: If a transaction T i has obtained an exclusive-mode lock (denoted by X) on item Q, then

T i can both read and write Q. We require that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. The transaction makes the request to the concurrency-control manager. The transaction can proceed with the operation only after the concurrency-control manager grants the lock to the transaction. The use of these two lock modes allows multiple transactions to read a data item but limits write access to just one transaction at a time.
The idea of locking is simple: when a transaction needs an assurance that some object, typically a database record that it is accessing in some way, will not change in some unpredictable manner while the transaction is not running on the CPU, it acquires a lock on that object. The lock prevents other transactions from accessing the object. Thus the first transaction can be sure that the object in question will remain in a stable state as long as the transaction desires. Several types of locks
are used in concurrency control: • Binary Locks • Shared/ Exclusive (or Read/Write)

Locks • Conversion of Locks Binary Locks A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested. We refer to the current value (or state) of the lock associated with item X as LOCK(X). Two operations, lock_item and
unlock_item,
are used with binary locking. A
149 transaction requests access to an item X by first issuing a lock_item(X) operation. If LOCK(X) = 1, the transaction is forced to wait. If LOCK(X) = 0, it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X. When the transaction is through using the item, it issues an unlock_item(X) operation, which sets LOCK(X) to 0 (unlocks the item) so that X may be accessed by other transactions. Hence, a binary lock enforces mutual exclusion on the data item.
It should be noted that
the lock_item and unlock_item operations must be implemented as indivisible units (atomic unit operations). If the simple binary locking scheme described here is used, every transaction must obey the following rules: • A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T. • A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T. • A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X. • A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X. Between the lock_item(X) and unlock_item(X) operations in transaction T, T is said to hold the lock on item X. At most one transaction can hold the lock on a particular item.
This locking system
is too restrictive for database items, because at most one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for reading purposes only. However, if a transaction is to write an item X, it must have exclusive access to X.
Shared/ Exclusive (or Read/Write)
Locks To recover from binary lock disadvantage, a different type of lock called a multiple-mode lock is used. In this scheme, called shared/ exclusive or read/
150 write locks, there are three locking operations: read_lock(
X),
write_lock(X), and unlock(X). A lock associated with an item X, LOCK(X), now has three possible states: "read-locked," "write-locked," or "unlocked." A read-locked item is also called share-locked, because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked, because a single transaction exclusively holds the lock on the item.
It should be noted that
each of the three operations should be considered indivisible. When we use the shared/exclusive locking scheme, the system must enforce the following rules: 1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T. 2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T. 3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T. 4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X. This rule may be relaxed. 5. A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X. This rule may be relaxed. 6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.
A compatibility function can be defined for the set of lock modes. Let A and B represent arbitrary lock modes. 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 transaction T i can be granted a lock on Q immediately, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B.
151 Such a function can be represented conveniently by a matrix. The compatibility relation between the two modes of locking appear in the matrix comp of figure 8.1. An element comp(A, B) of the matrix has the value true if and only if mode A is compatible with mode B. S X S true false X false false Figure 8.1 Lock-compatibility matrix comp. A transaction requests a shared lock on data item Q by executing the lock- S(Q) instruction. Similarly, a transaction requests an exclusive lock through the lock- X(Q) instruction. A transaction can unlock a data item Q by the unlock(Q) instruction.
Conversion of Locks If

a transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another. For example, it is possible for a transaction T to issue a read_lock(X) and then later on to upgrade the lock by issuing a write_lock(X) operation. If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded; otherwise, the transaction must wait. It is also possible for a transaction T to issue a write_lock(X) and then later on to downgrade the lock by issuing a read_lock(X) operation. When upgrading and downgrading of locks is used, the lock table must include transaction identifiers in the record structure for each lock (in the locking_transaction(s) field) to store the information on which transactions hold locks on the item. Using binary locks or read/write locks in transactions does not guarantee
152 serializability of schedules on its own.
To guarantee serializability, we must follow an additional protocol concerning the positioning of locking and unlocking operations in every transaction.
This best known protocol is two-phase locking. 8.2.1

98% **MATCHING BLOCK 150/202** W

Two-Phase Locking (2PL) One protocol that ensures serializability is the two-phase locking protocol. This protocol requires that each transaction issue lock and unlock requests in two phases: 1. Growing phase: A transaction may obtain locks, but may not release any lock. 2. Shrinking phase: A transaction may release locks, but may not obtain any new locks. Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

For example, transactions T 3 and T 4 are two phases. On the other hand, transactions T 1 and T 2 are not two phases. Note that the unlock instructions do not need to appear at the end of the transaction. For example, in the case of transaction T 3 , we could move the unlock(B) instruction to just after the lock- X (A) instruction, and still retain the two-phase locking property.

100% **MATCHING BLOCK 151/202** W

The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the lock point of the transaction.

Now, transactions can be ordered according to their lock points; this ordering is, in fact, a serializability ordering for the transactions. Two-phase locking may limit the amount of concurrency that can occur in a schedule. This is because a transaction T may not be able to release an item X after it is through using it if T must lock an additional item Y later on; or conversely, T must lock the additional item Y before it needs it so that it can release X. Hence, X must remain locked by T until all items that the transaction needs to read or write have been locked; only then can X be released by T.
153 Meanwhile, another transaction seeking to access X may be forced to wait, even though T is done with X; conversely, if Y is locked earlier than it is needed, another transaction seeking to access Y is forced to wait even though T is not using Y yet. This is the price for guaranteeing serializability of all schedules without having to check the schedules themselves.
We also know that in addition to being serializable, schedules should be cascadeless. Cascading rollback may occur under two-phase locking.

100% **MATCHING BLOCK 152/202** W

Cascading rollbacks can be avoided by a modification of two-phase locking called the strict two-phase locking protocol.

This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction is locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

100% **MATCHING BLOCK 153/202** W

Another variant of two-phase locking is the rigorous two-phase locking protocol, which requires that all locks be held until the transaction commits.

However, two-phase locking does not ensure freedom from deadlock.
If two transactions wait for locks on items held by the other, deadlock will occur and the deadlock detection and recovery scheme
will be
needed. It is also possible for transactions to be in livelock, that is, left in a wait state indefinitely, unable to acquire any new locks, although the DBMS is not in deadlock. This can happen if the waiting algorithm for transactions is unfair and does not take account of the time that transactions have been waiting. To avoid livelock, a priority system can be used, whereby the longer a transaction has to wait, the higher its priority, for example, a first-come-first-served queue can be used for waiting transactions. 8.3

100% **MATCHING BLOCK 154/202** W

Implementation of Locking A lock manager can be implemented as a process that receives messages from transactions and sends messages in reply. The lock-manager process replies to lock-request messages with lock-grant messages, or with messages requesting 154 rollback of the transaction (

in case of deadlocks).

100% **MATCHING BLOCK 155/202** W

Unlock messages require only an acknowledgment in response, but may result in a grant message to another waiting transaction. The lock manager uses this data structure: For each data item that is currently locked, it maintains a linked list of records, one for each request, in the order in which the requests arrived. It uses a hash table, indexed on the name of a data item, to find the linked list (if any) for a data item; this table is called the lock table. Each record of the linked list for a data item notes which transaction made the request, and what lock mode it requested. The record also notes if the request has currently been granted.

Figure 8.2 shows an example of a lock table. The table contains locks for five different data items, I4, I7, I23, I44, and I912. The lock table uses overflow chaining, so there is a linked list of data items for each entry in the lock table. There is also a list of transactions that have been granted locks, or are waiting for locks, for each of the data items. Granted locks are the rectangles filled in a darker shade, while waiting requests are the rectangles filled in a lighter shade. Figure 8.2 Lock Table 155

98% **MATCHING BLOCK 156/202** W

The lock manager processes requests this way: • When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present. Otherwise it creates a new linked list, containing only the record for the request. It always grants a lock request on a data item that is not currently locked. But if the transaction requests a lock on an item on which a lock is currently held, the lock manager grants the request only if it is compatible with the locks that are currently held, and all earlier requests have been granted already. Otherwise the request has to wait. • When the lock manager receives an unlock message from a transaction, it deletes the record for that data item in the linked list corresponding to that transaction. It tests the record that follows, if any, to see if that request can now be granted. If it can, the lock manager grants that request, and processes the record following it, if any, similarly, and so on. • If a transaction aborts, the lock manager deletes any waiting request made by the transaction. Once the database system has taken appropriate actions to undo the transaction, it releases all locks held by the aborted transaction.

This algorithm guarantees freedom from starvation for lock requests, since a request can never be granted while a request received earlier is waiting to be granted.
Starvation occurs when the same transaction is always chosen as the victim, and the transaction can never complete. The DBMS can avoid starvation by storing a count of the number of times a transaction has been selected as the victim and using a different selection criterion once this count reaches some upper limit. 8.4
Granularity of Data Items Another factor that affects concurrency control is the granularity of the data items. It is a portion of the database that a data item represents. An item can be
156 as small as a single attribute (field) value or as large as a disk block, or even a whole file or the entire database.

All the concurrency control protocols assume that the database consists of a number of “data items,” without explicitly defining the term. Typically, a data item ranges from coarse to fine, where fine granularity refers to small item sizes and coarse granularity refers to large item sizes. The
data items can be chosen from
the entire database; a file; a page (sometimes called an area or database space—a section of physical disk in which relations are stored); a record; a field value of a record. The size or granularity of the data item that can be locked in a single operation has a significant effect on the overall performance of the concurrency control algorithm. However, there are several tradeoffs that have to be considered in choosing the data item size.
For example,
consider a transaction that updates a single tuple of a relation. The concurrency control algorithm might allow the transaction to lock only that single tuple, in which case the granule size for locking is a single record. On the other hand, it might lock the entire database, in which case the granule size is the entire database. In the second case, the granularity would prevent any other transactions from executing until the lock is released. This would clearly be undesirable. On the other hand, if a transaction updates 95% of the records in a file, then it would be more efficient to allow it to lock the entire file rather than to force it to lock each record separately. However, escalating the granularity from field or record to file may increase the likelihood of deadlock occurring. Thus, the coarser the data item size, the lower the degree of concurrency permitted. On the other hand, the finer the item size, the more locking information that needs to be stored. The best item size depends upon the nature of the transactions. If a typical transaction accesses a small number of records, it is advantageous to have the data item granularity at the record level. On the other hand, if a transaction typically accesses many records of the same file, it may be better to have page or file granularity so that the transaction considers all those records as
157 one
data item. There are some techniques that have been proposed for
dynamic data item sizes. With these techniques, depending on the types of transaction that are currently executing, the data item size may be changed to the granularity that best suits these transactions. Ideally, the DBMS should support mixed granularity with record, page, and file-level locking. Some systems automatically upgrade locks from record or page to file if a particular transaction is locking more than a certain percentage of the records or pages in the file. The granularity of locks can be represented in a hierarchical structure where each node represents data items of different sizes, as shown in figure 8.3. Here, the root node represents the entire database, the level 1 nodes represent files, the level 2 nodes represent
pages, the level 3 nodes represent records, and the level 4 leaves represent individual fields. Whenever a node is locked, all its descendants are also locked. For example, if a transaction locks a page, Page 2, all its records (Record 1 and Record 2) as well as all their fields (Field 1 and Field 2) are also locked.
158
Figure 8.3 Different levels of Locking
If another transaction requests an incompatible lock on the same node, the DBMS clearly knows that the lock cannot be granted. If another transaction requests a lock on any of the descendants of the locked node, the DBMS checks the hierarchical path from the root to the requested node
to determine whether any of its ancestors are locked before deciding whether to grant the lock. Thus, if the request is for an exclusive lock on record Record 1, the DBMS checks its parent (Page 2), its grandparent (File 2), and the database itself to determine whether any of them are locked. When it finds that Page 2 is already locked, it denies the request. Additionally, a transaction may request a lock on a node and a descendant of the node is already locked. For example, if a lock is requested on File 2, the DBMS checks every page in the file, every record in those pages, and every field in those records to determine whether any of them are locked.
159 8.4.1
Multiple-Granularity Locking To reduce the searching involved in locating locks on descendants, the DBMS can use another specialized locking strategy called multiple-granularity locking. This strategy uses a new type of lock called an intention lock. When any node is locked, an intention lock is placed on all the ancestors of the node. Thus, if some descendant of File 2 (in our example, Page 2) is locked and a request is made for a lock on File 2 , the presence of an intention lock on File 2 indicates that some descendant of that node is already locked.
Intention locks may be either shared (
S) or exclusive (X).

An intention shared (IS) lock conflicts only with an exclusive lock; an intention exclusive (IX) lock conflicts with both a shared and an exclusive lock. In addition, a transaction can hold a shared and intention exclusive (SIX) lock that is logically equivalent to holding both a shared and an IX lock. A SIX lock conflicts with any lock that conflicts with either a shared or IX lock; in other words, a SIX lock is compatible only with an IS lock. The lock compatibility table for multiple-granularity locking is shown in figure 8.4. To ensure serializability with locking levels, a two-phase locking protocol is used as follows: • No lock may be granted once any node has been unlocked. • No node may be locked until its parent is locked by an intention lock. • No node may be unlocked until all its descendants are unlocked.
Figure 8.4 Lock compatibility table for multiple-granularity locking.
160
In this way, locks are applied from the root down using intention locks until the node requiring an actual read or exclusive lock is reached, and locks are released from the bottom up. However, deadlock is still possible and must be handled as discussed previously.

88% **MATCHING BLOCK 157/202** W

The multiple-granularity locking protocol uses these lock modes to ensure serializability. It requires that a transaction T i that attempts to lock a node Q must follow these rules: 1. Transaction T i must observe the lock-compatibility function of Figure 8.4. 2. Transaction T i must lock the root of the tree first, and can lock it in any mode. 3. Transaction T i can lock a node Q in S or IS mode only if T i currently has the parent of Q locked in either IX or IS mode. 4. Transaction T i can lock a node Q in X, SIX , or IX mode only if T i currently has the parent of Q locked in either IX or SIX mode. 5. Transaction T i can lock a node only if T i has not previously unlocked any node (that is, T i is two phase). 6. Transaction T i can unlock a node Q only if T i currently has none of the children of Q locked.

It should be observed

100% **MATCHING BLOCK 158/202** W

that the multiple-granularity protocol requires that locks be acquired in top-down (root-to-leaf) order, whereas locks must be released in bottom-up (leaf-to-root) order. 8.5

Concurrency control in index structures using locks Two-phase locking can also be applied to indexes, where the nodes of an index correspond to disk pages. However, holding locks on index pages until the shrinking phase of 2PL could cause an undue amount of transaction blocking. This is because searching an index always starts at the root, so if a transaction
161 wants to insert a record (write operation), the root would be locked in exclusive mode, so all other conflicting lock requests for the index must wait until the transaction enters its shrinking phase. This blocks all other transactions from accessing the index, so in practice other approaches to locking an index must be used. The tree structure of the index can be taken advantage of when developing a concurrency control scheme. For example, when an index search (read operation) is being executed, a path in the tree is traversed from the root to a leaf. Once a lower-level node in the path has been accessed, the higher-level nodes in that path will not be used again.
So once a read lock on
a child node is obtained, the lock on the parent can be released.
Second, when an insertion is being applied to a leaf node (that is, when a key and a pointer are inserted), then a specific leaf node must be locked in exclusive mode. However, if that
node is not full, the insertion will not cause changes to higher-level
index nodes, which implies that they need not be locked exclusively. Another approach to index locking is to use a variant of the B + -tree, called the B-link tree. In a B-link tree, sibling nodes on the same level are linked together at every level. This allows shared locks to be used when requesting a page and requires that the lock be released before accessing the child node. For an insert operation, the shared lock on a node would be upgraded to exclusive mode. If a split occurs, the parent node must be relocked in exclusive mode. One complication is for search operations executed concurrently with the update. Suppose that a concurrent update operation follows the same path as the search, and inserts a new entry into the leaf node. In addition, suppose that the insert causes that leaf node to split. When the insert is done, the search process resumes, following the pointer to the desired leaf, only to find that the key it is looking for is not present because the split has moved that key into a new leaf node, which would be the right sibling of the original leaf node. However, the search process can still succeed if it follows the pointer (link) in the original leaf node to its right sibling, where the desired key has been moved.

162 8.6 Other Concurrency Control Issues In this section, we will discuss the different issues relevant to concurrency control. The problems associated with insertion and deletion of records and the so-called phantom problem, is discussed. 8.6.1 Insertion, Deletion, and Phantom Records When a new data item is inserted in the database, it obviously cannot be accessed until after the item is created and the insert operation is completed. In a locking environment, a lock for the item can be created and set to exclusive (write) mode; the lock can be released at the same time as other write locks would be released, based on the concurrency control protocol being used. For a timestamp-based protocol, the read and write timestamps of the new item are set to the timestamp of the creating transaction. A deletion operation is applied on an existing data item. For locking protocols, again an exclusive (write) lock must be obtained before the transaction can delete the item. For timestamp ordering, the protocol must ensure that no later transaction has read or written the item before allowing the item to be deleted. A situation known as the phantom problem can occur when a new record that is being inserted by some transaction T satisfies a condition that a set of records accessed by another transaction T must satisfy. For example, suppose that transaction T is inserting a new EMPLOYEE record whose DNO = 5, while transaction T′ is accessing all EMPLOYEE records whose DNO = 5 (say, to add up all their SALARY values to calculate the personnel budget for department 5). If the equivalent serial order is T followed by T′, then T must read the new EMPLOYEE record and include its SALARY in the sum calculation. For the equivalent serial order T′ followed by T, the new salary should not be included. It should be noted that although the transactions logically conflict, in the latter case there is really no record (data item) in common between the two transactions, since T may have locked all the records with DNO = 5 before T inserted the new record. This is because the record that causes the conflict is a
163 phantom record that has suddenly appeared in the database on being inserted. If other operations in the two transactions conflict, the conflict due to the phantom record may not be recognized by the concurrency control protocol. One solution to the phantom record problem is to use index locking, as an index includes entries that have an attribute value, plus a set of pointers to all records in the file with that value. A more general technique, called predicate locking, would lock access to all records that satisfy an arbitrary predicate (condition) in a similar manner; however predicate locks have proved to be difficult to implement efficiently. 8.6.2 Interactive Transactions Another problem occurs when interactive transactions read input and write output to an interactive device, such as a monitor screen, before they are committed. The problem is that a user can input a value of a data item to a transaction T that is based on some value written to the screen by transaction T, which may not have committed. This dependency between T and T′ cannot be modeled by the system concurrency control method, since it is only based on the user interacting with the two transactions. An approach to dealing with this problem is to postpone output of transactions to the screen until they have committed. 8.6.3 Latches Locks held for a short duration are typically called latches. Latches do not follow the usual concurrency control protocol such as two-phase locking. For example, a latch can be used to guarantee the physical integrity of a page when that page is being written from the buffer to disk. A latch would be acquired for the page, the page written to disk, and then the latch is released. 8.7 Summary •
A lock is a variable associated with a data item
in the database and
164 describes the status of that data item
with respect to possible operations that can be applied to the item. •

97% **MATCHING BLOCK 160/202** W

A locking protocol is a set of rules that state when a transaction may lock and unlock each of the data items in the database. • The two-phase locking protocol allows a transaction to lock a new data item only if that transaction has not yet unlocked any data item. The protocol ensures serializability, but not deadlock freedom. In the absence of information concerning the manner in which data items are accessed, the two-phase locking protocol is both necessary and sufficient for ensuring serializability. •

A tree may be used to represent the granularity of locks in a system that allows locking of data items of different sizes. When an item is locked, all its descendants are also locked. When a new transaction requests a lock, it is easy to check all the ancestors of the object to determine whether they are already locked. To show whether any of the node's descendants are locked, an intention lock is placed on all the ancestors of any node being locked. 8.8
Key Terms •

91% **MATCHING BLOCK 161/202** W

Shared: If a transaction T i has obtained a shared-mode lock (denoted by S) on item Q, then

T i

93% **MATCHING BLOCK 162/202** W

can read, but cannot write, Q. • Exclusive: If a transaction T i has obtained an exclusive-mode lock (denoted by X) on item Q, then

T i can both read and write Q. •

100% **MATCHING BLOCK 163/202** W

Growing phase: A transaction may obtain locks, but may not release any lock. • Shrinking phase: A transaction may release locks, but may not obtain any new locks. •

94% **MATCHING BLOCK 164/202** W

Lock point: The point in the schedule where the transaction has obtained 165 its final lock (the end of its growing phase) is called the lock point of the transaction. •

Starvation: Starvation occurs when the same transaction is always chosen as the victim, and the transaction can never complete. 8.9
Check Your Progress Short- Answer Type Q1) Shared/ Exclusive locks have _____ locking operations. Q2) A situation known as the _____ can occur when a new record that is being inserted by some transaction T satisfies a condition that a set of records accessed by another transaction T must satisfy. Q3) Using binary locks or read/write locks in transactions does not guarantee serializability of schedules on its own. (True/ False?) Q4)
Fine granularity
refers to _____ item sizes and coarse granularity refers to _____item sizes.
Q5) Which of the following locks are used by Multiple-granularity locking? a) Binary locks b) Share locks c) Intention locks d) None Long- Answer Type Q1) What is the two-phase locking protocol? How does it guarantee serializability? Q2) Compare binary locks to exclusive/shared locks. Why is the latter type of locks preferable? Q3) What is multiple granularity locking? Under what circumstances is it used? Q4) What is a phantom record? Discuss the problem that a phantom record can cause for concurrency control.
166 Q5) How does the granularity of data items affect the performance of concurrency control? What factors affect selection of granularity size for data items? References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011. •
Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007.
167 MODULE- V OBJECT- ORIENTED DBMS
168
169
Unit: 09
Object-Oriented DBMS: Concept & Design Structure 9.0 Introduction 9.1 Unit Objectives 9.2 Overview of Object-oriented paradigm 9.3 Object Identity & Object Structure 9.4 Type hierarchies and inheritance 9.5 Object-Oriented Data Models 9.6 Persistent programming languages 9.7 Summary 9.8 Key Terms 9.9 Check Your Progress 9.0 Introduction The traditional data models and systems, such as relational, network, and hierarchical, have been quite successful in developing the database technology required for many traditional business database applications. However, they have certain shortcomings when more complex database applications must be designed and implemented. Object-oriented databases were proposed to meet the needs of these more complex applications. The object-oriented approach offers the flexibility to handle some of these requirements without being limited by the data types and query languages available in traditional database systems. A key feature of object-oriented databases is the power they give the designer to specify both the structure of complex objects and the operations that can be applied to these objects. Another reason for the creation of object-oriented databases is the increasing use

170 of object-oriented programming languages in developing software applications. Databases are now becoming fundamental components in many software systems, and traditional databases are difficult to use when embedded in object- oriented software applications that are developed in
an object-oriented programming language such as C++ or JAVA.
Object-oriented databases are designed so they can be directly or seamlessly integrated with software that is developed using object-oriented programming languages. This unit focuses on the basic architecture of object-oriented DBMS. 9.1 Unit Objectives After completing this unit, the reader will be able to: • Have a basic idea about Object-oriented DBMS (OODBMS). • Learn about the hierarchies and inheritance of OODBMS. • Discuss the framework of Object- oriented data model. • Learn
the basics of the functional data model and persistent programming languages. 9.2
Overview of Object-oriented paradigm
In the late 1960s and early 1970s, there were two mainstream approaches to constructing DBMSs. The first approach was based on the hierarchical data model from IBM, in response to the enormous information storage requirements. The second approach was based on the network data model, which attempted to create a database standard and resolve some of the difficulties of the hierarchical model, such as its inability to represent complex relationships effectively. Together, these approaches represented the first generation of DBMSs. Although these two approaches had some disadvantages,
such as,
complex programs had to be written to answer even simple queries based on navigational
171 record-oriented access; there was minimal data independence; there was no widely accepted theoretical foundation. In 1970,
these disadvantages were addressed by the approach based on relational data models. This approach was referred to as the second generation of DBMSs. Relational DBMS is discussed in upcoming units.
In response to the increasing complexity of database applications, two new data models have emerged: the Object-Oriented Data Model (OODM) and the Object- Relational Data Model (ORDM).
This evolution represents the third-generation of DBMSs. Figure 9.1 History of Data Models Different researchers have proposed different definitions and prospectus of the third generation data models. •
Kim (1991) defines an Object-Oriented Data Model (OODM) as
a (logical) data model that captures the semantics of objects supported in object-
172 oriented programming;
Object-Oriented Database (OODB) as
a persistent and sharable collection of objects defined by an OODM;
and an Object- Oriented DBMS (OODBMS) as the manager of an OODB. •
Zdonik and Maier (1990) present a threshold model that an OODBMS must, at a minimum, satisfy: 1. It must provide database functionality. 2.
It must support object identity. 3. It must provide encapsulation. 4. It must support objects with complex state. ●
Khoshafian and Abnous (1990) define an OODBMS as: 1. Object-orientation = abstract data types + inheritance + object identity; 2. OODBMS = object-orientation + database capabilities. ●
Parsaye et al. (1989) defines OODBMS as : 1.
high-level query language with query optimization capabilities in the underlying system; 2. support for persistence, atomic transactions, and concurrency and recovery control; 3. support for complex object storage, indexes, and access methods for fast and efficient retrieval; 4. OODBMS = object-oriented system + (1) + (2) + (3).
173 Figure 9.2 Origins of
the
object-oriented data model. 9.3 Object Identity & Object Structure An object typically has two components: state (value) and behavior (operations). Hence, it is somewhat similar to a program variable in a programming language, except that it will typically have a complex data structure as well as specific operations defined by the programmer. Objects in an Object Oriented Programming Language (OOPL) exist only during program execution and are hence called transient objects.
An Object-Oriented database can extend the existence of objects so that they are stored permanently,
and hence
the objects persist beyond program termination and can be retrieved later and shared by other programs.
In other words, Object-Oriented databases store persistent objects permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications. This requires the incorporation of other well-known features of database management systems, such as indexing mechanisms, concurrency control, and recovery. An OODB system interfaces with one or more Object-Oriented programming languages to provide persistent and

174 shared object capabilities. OODBs provide a unique system-generated object identifier (OID) for each object. One goal of OODBs is to maintain
a direct correspondence between real- world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon. Object Identity An OODB system provides a unique identity to each independent object stored in the database. This unique identity is typically implemented via a unique, system- generated object identifier. The value of an OID is not visible to the external user, but it is used internally by the system to identify each object uniquely and to create and manage inter-object references. The main property required of an OID is that it be immutable, that means the OID value of a particular object should not change. This preserves the identity of the real-world object being represented. Hence, an OODB system must have some mechanism for generating OIDs and preserving the immutability property. Another property is that each OID is used only once; that is, even if an object is removed from the database, its OID should not be assigned to another object. These two properties imply that the OID should not depend on any attribute values of the object, since the value of an attribute may be changed or corrected. Some early OODMs required that everything from a simple value to a complex object should be represented as an object; hence, every basic value, such as an integer, string, or Boolean value, has an OID. This allows two basic values to have different OIDs, which can be useful in some cases. For example, the integer value 100 can be used sometimes to mean a weight in kilograms and at other times to mean the age of a person. Then, two basic objects with distinct OIDs could be created, but both objects would represent the integer value 100. Hence, most OODB systems allow for the representation of both objects and values. Every object must have an immutable OID, whereas a value has no OID and just stands for itself. Hence, a value is typically stored within an object and cannot be
175 referenced from other objects. In some systems, complex structured values can also be created without having a corresponding OID if needed. Object Structure In OODBs, the state (current value) of a complex object may be constructed from other objects (or other values) by using certain type constructors. One formal way of representing such objects is to view each object as a triple (i, c, v), where i is a unique object identifier (the OID), c is a type constructor (that is, an indication of how the object state is constructed), and v is the object state (or current value). The data model will typically include several type constructors. The three most basic constructors are atom, tuple, and set. Other commonly used constructors include list, bag, and array. The atom constructor is used to represent all basic atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly. The object state v of an object (i, c, v) is interpreted based on the constructor c. • If c = atom, the state (value) v is an atomic value from the domain of basic values supported by the system. • If c= set, the state v is a set of object identifiers, which are the OIDs for a set of objects that are typically of the same type. • If c = tuple, the state v is a tuple of the form, where each is an attribute name and each is an OID. • If c = list, the value v is an ordered list of OIDs of objects of the same type. A list is similar to a set except that the OIDs in a list are ordered, and hence we can refer to the first, second, or object in a list. • For c = array, the state of the object is a single-dimensional array of object identifiers. The main difference between array and list is that a list can have an arbitrary number of elements whereas an array typically has a maximum size. The
176 difference between set and bag is that all elements in a set must be distinct whereas a bag can have duplicate elements. This model of objects allows arbitrary nesting of the set, list, tuple, and other constructors. The state of an object that is not of type atom will refer to other objects by their object identifiers. Hence, the only case where an actual value appears is in the state of an object of type atom. The type constructors set, list, array, and bag are called collection types (or bulk types), to distinguish them from basic types and tuple types. The main characteristic of a collection type is that the state of the object will be a collection of objects that may be unordered (such as a set or a bag) or ordered (such as a list or an array). The tuple type constructor is often called a structured type, since it corresponds to the struct construct in the C and C++ programming languages. 9.4 Type Hierarchies and Inheritance Another main characteristic of Object-oriented database systems is that they allow type hierarchies and inheritance. Type hierarchies in databases usually imply a constraint on the extents corresponding to the types in the hierarchy. A different Object-Oriented model is considered in which attributes and operations are treated uniformly, since both attributes and operations can be inherited. In most database applications, there are numerous objects of the same type or class. Hence, OODBs must provide a capability for classifying objects based on their type, as do other database systems. But in OODBs, a further requirement is that the system permits the definition of new types based on other predefined types, leading to a type (or class) hierarchy. Typically, a type is defined by assigning it a type name and then defining a number of attributes (instance variables) and operations (methods) for the type. In some cases, the attributes and operations are together called functions, since attributes resemble functions with zero arguments. A function name can be used to refer to the value of an attribute or to refer to the resulting value of an

177 operation (method). Here, we use the term function to refer to both attributes and operations of an object type, since they are treated similarly in a basic introduction to inheritance. A type in its simplest form can be defined by giving it a type name and then listing the names of its visible (public) functions. When specifying a type, following format is used, which does not specify arguments of functions, to simplify the discussion: TYPE_NAME: function, function, . . . , function For example, a type that describes characteristics of a PERSON may be defined as follows: PERSON: Name, Address, Birthdate, Age In the PERSON type, the Name, Birthdate and Address functions can be implemented as stored attributes, whereas the Age function can be implemented as a method that calculates the Age from the value of the Birthdate attribute and the current date. The concept of subtype is useful when the designer or user must create a new type that is similar but not identical to an already defined type. The subtype then inherits all the functions of the predefined type, which we shall call the supertype. For example, suppose that we want to define two new types EMPLOYEE and STUDENT as follows: EMPLOYEE: Name, Address, Birthdate, Age, Salary, HireDate, Seniority STUDENT: Name, Address, Birthdate, Age, Major, GPA Since both STUDENT and EMPLOYEE include all the functions defined for PERSON plus some additional functions of their own, we can declare them to be subtypes of PERSON. Each will inherit the previously defined functions of PERSON —namely, Name, Address, Birthdate, and Age. For STUDENT, it is only necessary to define the new (local) functions Major and GPA, which are not inherited. Presumably, Major can be defined as a stored attribute, whereas GPA 178 may be implemented as a method that calculates the student's grade point average by accessing the Grade values that are internally stored (hidden) within each STUDENT object as private attributes. For EMPLOYEE , the Salary and HireDate functions may be stored attributes, whereas Seniority may be a method that calculates Seniority from the value of HireDate. The idea of defining a type involves defining all of its functions and implementing them either as attributes or as methods. When a subtype is defined, it can then inherit all of these functions and their implementations. Only functions that are specific or local to the subtype, and hence are not implemented in the supertype, need to be defined and implemented. Therefore, we can declare EMPLOYEE and STUDENT as follows: EMPLOYEE subtype-of PERSON: Salary, HireDate, Seniority STUDENT subtype-of PERSON: Major, GPA In general, a subtype includes all of the functions that are defined for its supertype plus some additional functions that are specific only to the subtype. Hence, it is possible to generate a type hierarchy to show the supertype/subtype relationships among all the types declared in the system. Type Extents In most OODBs, the collection of objects in an extent has the same type or class. However, this is not a necessary condition. For example, SMALLTALK, a so-called typeless Object-Oriented language, allows a collection of objects to contain objects of different types. This can also be the case when other non-object-oriented typeless languages, such as LISP, are extended with Object-Oriented concepts. However, since the majority of OODBs support types, we will assume that extents are collections of objects of the same type for the remainder of this section. It is common in database applications that each type or subtype will have an extent associated with it, which holds the collection of all persistent objects of that type or subtype. In this case, the constraint is that every object in an extent 179 that corresponds to a subtype must also be a member of the extent that corresponds to its supertype. Some OODB systems have a predefined system type (called the ROOT class or the OBJECT class) whose extent contains all the objects in the system. Classification then proceeds by assigning objects into additional subtypes that are meaningful to the application, creating a type hierarchy or class hierarchy for the system. All extents for system and user-defined classes are subsets of the extent corresponding to the class OBJECT , directly or indirectly. In most Object-Oriented systems, a distinction is made between persistent and transient objects and collections. A persistent collection holds a collection of objects that is stored permanently in the database and hence can be accessed and shared by multiple programs. A transient collection exists temporarily during the execution of a program but is not kept when the program terminates. For example, a transient collection may be created in a program to hold the result of a query that selects some objects from a persistent collection and copies those objects into the transient collection. The transient collection holds the same type of objects as the persistent collection. The program can then manipulate the objects in the transient collection, and once the program terminates, the transient collection ceases to exist. In general, numerous collections—transient or persistent—may contain objects of the same type. 9.5 Object-Oriented Data Models A basic functional data model of the family of semantic data models is discussed here. It is the simplest and interesting
data model (Kerschberg and Pacheco, 1976;
Sibley and Kerschberg, 1977). This model shares certain ideas with the object approach, including object identity, inheritance, overloading, and navigational access. In
this data model,
any data retrieval task can be viewed as the process of evaluating and returning the result of a function with zero, one, or more arguments. The resulting data model is conceptually simple
yet also
very

180 expressive. The main modeling primitives are entities and functional relationships. Entities Entities are decomposed into (abstract) entity types and printable entity types. Entity types correspond to classes of real-world objects and are declared as functions with zero arguments that return the type ENTITY. For example, we could declare the Staff and PropertyForRent entity types as follows: Staff () →ENTITY PropertyForRent () → ENTITY Printable entity types are analogous to the base types in a programming language and include: INTEGER, CHARACTER, STRING, REAL, and DATE. An attribute is defined as
a functional
relationship, taking the entity type as an argument and returning a printable entity type. Some of the attributes of the Staff entity type could be declared as follows: staffNo(Staff) → STRING sex(Staff) → CHAR salary(Staff) → REAL Thus, applying the function staffNo to an entity of type Staff returns that entity's staff number, which is a printable value of type STRING. We can declare a composite attribute by first declaring the attribute to be an entity type and then declaring its components as functional relationships of the entity type. For example, we can declare the composite attribute Name of Staff as follows: Name() → ENTITY Name(Staff) → NAME fName(Name) → STRING lName(Name) → STRING
181 Relationships Functions with arguments model not only the properties (attributes) of entity types but also relationships between entity types. Thus, the data model makes no distinction between attributes and relationships. Each relationship may have an inverse relationship defined. For example, we may model the one-to-many relationship Staff Manages PropertyForRent as follows: Manages(Staff) PropertyForRent ManagedBy(PropertyForRent) ↠ Staff INVERSE OF Manages In this example, the double-headed arrow is used to represent a one-to-many relationship. This notation can also be used to represent multi-valued attributes.
Many-to-many relationships can be modeled by using the double- headed arrow in both directions. For example, we may model the *.* relationship Client Views PropertyForRent as follows: Views(Client) ↠ PropertyForRent ViewedBy(PropertyForRent) ↠ Client INVERSE OF Views
It should be noted
that
an entity (instance) is some form of token identifying a unique object in the database and typically representing a unique object in the real world. In addition, a function maps a given entity to one or more target entities (for example, the function Manages maps a particular Staff entity to a set of PropertyForRent entities). Thus, all
inter object
relationships are modeled by associating the corresponding entity instances and not their names or keys. Thus, referential integrity is an implicit part of the functional data model and requires no explicit enforcement, unlike the relational data model. The data model also supports multi-valued functions. For example, we can model the attribute viewDate of the previous relationship Views as follows: viewDate(Client, PropertyForRent) → DATE
182 Inheritance and path expressions The data model supports inheritance through entity types. For example, the function Staff () returns a set of staff entities formed as a subset of the ENTITY type. Thus, the entity type Staff is a subtype of the entity type ENTITY. This subtype/ supertype relationship can be extended to any level. As would be expected, subtypes inherit all the functions defined over all of its supertypes. The data model also supports the principle of substitutability, so that an instance of a subtype is also an instance of its supertypes. For example, we could declare the entity type Supervisor to be a subtype of the entity type Staff as follows: Staff() → ENTITY Supervisor() → ENTITY IS-A-STAFF(Supervisor) → Staff The data model allows derived functions to be defined from the composition of multiple functions. Thus, we can define the following derived functions (note the overloading of function names): fName(Staff) → fName(Name(Staff)) fName(Supervisor) → fName(IS-A-STAFF(Supervisor)) The first derived function returns the set of first names of staff by evaluating the composite function on the right-hand side of the definition. Following on from this, in the second case the right-hand side of the definition is evaluated as the composite function fName(Name(IS-A-STAFF(Supervisor))). This composition is called a path expression and may be more recognizable written in dot notation: Supervisor.lS-A-STAFF.Name.fName
There have been many proposals for functional data models and languages. The two earliest were FQL (Buneman and Frankel, 1979) and perhaps the best known DAPLEX (Shipman, 1981). The attraction of the functional style of these languages has produced many systems such as GDM (Batory et al., 1988), the Extended FDM (Kulkarni and Atkinson, 1986, 1987), FDL (Poulovassilis and King,
183 1990), PFL (Poulovassilis and
Small, 1991), and P/ FDM (Gray et al., 1992). The functional data languages have also been used with
non-functional
data models, such as PDM (Manola and Dayal, 1986), IPL (Annevelink, 1991), and LIFOO (Boucelma and Le Maitre, 1991).
9.6
Persistent Programming Languages Another interesting and separate area of development in OODBMs is persistent programming languages. It is

a language that provides its users with the ability to (transparently) preserve data across successive executions of a program and even allows such data to be used by many different programs. Data in a persistent programming language is independent of any program, able to exist beyond the execution, and lifetime of the code that created it. Such languages were originally intended to provide neither full database functionality nor access to data from multiple languages (Cattell, 1994).
A
database programming language is
a language that integrates some ideas from the database programming model with traditional programming language features. In contrast, a database programming language is distinguished from a persistent programming language by its incorporation of features beyond persistence, such as transaction management, concurrency control, and recovery (Bancilhon and Buneman, 1990). The ISO SQL standard specifies that SQL can be embedded in the programming languages C, Fortran, Pascal, COBOL, Ada, MUMPS, and PL/1. Communication is through a set of variables in the host language, and a special preprocessor modifies the source code to replace the SQL statements with calls to DBMS routines. The source code can then be compiled and linked in the normal way. Alternatively, an API can be provided, removing the need for any precompilation. Although the embedded approach is rather clumsy, it was useful and necessary, as the SQL2 standard was not computationally complete. The problems with using two different language paradigms have been collectively
184 called the impedance mismatch between the application programming language and the database query language. Researchers working on the development of persistent programming languages have been motivated primarily by the following aims (Morrison et al., 1994): • improving programming productivity by using simpler semantics; • removing ad hoc arrangements for data translation and long-term data storage; • providing protection mechanisms over the whole environment. Persistent programming languages attempt to eliminate the impedance mismatch by extending the programming language with database capabilities. In a persistent
programming
language, the language's type system provides the data model, which usually contains rich structuring mechanisms. In some languages like PS-algol and Napier88, procedures are "first-class" objects and are treated like any other data objects in the language. For example, procedures are assignable, may be the result of expressions, other procedures or blocks, and may be elements of constructor types. Among other things, procedures can be used to implement abstract data types. The act of importing an abstract data type from the persistent store and dynamically binding it into a program is equivalent to module-linking in more traditional languages. The second important aim of a persistent programming language is to maintain the same data representation in the application memory space as in the persistent
store on secondary storage. This overcomes the difficulty and overhead of mapping between the two representations. The addition of (transparent) persistence into a programming language is an important enhancement to an interactive development environment, and the integration of the two paradigms provides increased functionality and semantics. The research into persistent programming languages has had a significant influence on the development of OODBMSs, and many of the issues
in
OODBMSs. The more encompassing term
Persistent Application System (PAS)
185 is sometimes used now instead of persistent programming language (Atkinson and Morrison, 1995). 9.7
Summary • 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 Object-Relational Data Model (ORDM). •
An OODBMS is a manager of an OODB. An OODB is a persistent and shared repository of objects defined in an OODM. An OODM is a data model that captures the semantics of objects supported in object-oriented programming. •
There
are two types of OID: logical OIDs, which
are independent of the physical location of the object on disk, and physical OIDs, which encode the location. In the former case, a level of indirection is required to look up the physical address of the object on disk. In both cases, however, an OID is different in size from a standard in-memory pointer, which need be
only large enough to address all virtual memory. •

The functional data model (FDM) shares certain ideas with the object approach, including object identity, inheritance, overloading, and navigational access. In the FDM, any data retrieval task can be viewed as the process of evaluating and returning the result of a function with zero, one, or more arguments. In the FDM, the main modeling primitives are entities (either entity types or printable entity types) and functional relationships. • A persistent programming language is a language that provides its users with the ability to (transparently) preserve data across successive executions of a program. Data in a persistent programming language is
186 independent of any program, able to exist beyond the execution and lifetime of the code that created it. However, such languages were originally intended to provide neither full database functionality nor access to data from multiple languages. 9.8
Key Terms • Object identity: Objects have unique identities that are independent of their attribute values. • Type constructors: Complex object structures can be constructed by recursively applying a set of basic constructors, such as tuple, set, list, and bag. • Type hierarchies and inheritance: Object types can be specified by using a type hierarchy, which allows the inheritance of both attributes and methods of previously defined types. • Extents: All persistent objects of a particular type can be stored in an extent. Extents corresponding to a type hierarchy have set/subset constraints enforced on them. 9.9 Check Your Progress Short- Answer Type Q1) _____ is the set of all instances of a class within a database. Q2)
Which of the following is an atomic literal? a) String b) Boolean c) Long d) All of the above
Q3) OODBMS has the ability to store complex data types on the Web. (True/ False?) Q4) _____ is an unordered collection of elements that may contain duplicates.
187 Q5) Which of the following is an ordered collection of elements of the same type? a) Set b) List c) Bag d) None of the above Long- Answer Type Q1) Discuss the various type constructors. How are they used to create complex object structures? Q2) What primary characteristics should an OID possess? Q3) State the difference between persistent and transient objects. Q4)
Describe the main modeling component of the functional data model.
Q5) Describe the
three generations of DBMSs. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007.
188 Unit: 10 Other Concepts of Object-Oriented Databases Structure 10.0 Introduction 10.1 Unit Objectives 10.2 Object-oriented database design 10.3 Persistence in OODBMSs 10.4 Encapsulation 10.5 Complex Objects 10.6 Issues in OODBMS 10.7 Advantages and Disadvantages of OODBMSs 10.8 Summary 10.9 Key Terms 10.10 Check Your Progress
10.0 Introduction We are already aware about the basics of object-oriented database systems. Today Object-Oriented concepts are applied in the areas of databases, software engineering, knowledge bases, artificial intelligence, and computer systems in general. OOPLs have their roots in the SIMULA language, which was proposed in the late 1960s. In SIMULA, the concept of a class groups together the internal data structure of an object in a class declaration. Subsequently, researchers proposed the concept of abstract data type, which hides the internal data structures and specifies all possible external operations that can be applied to an object, leading to the concept of encapsulation. In this unit we will discuss how to adapt the methodology of an OODBMS. Some other concepts of OODBs like encapsulation, complex objects, and persistence
189 are also discussed. This unit also focuses on the issues faced in OODBMS and advantages & disadvantages of OODBMSs. 10.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the main strategies for developing an OODBMS. • Discuss the different schemes for providing persistence in programming languages. • Illustrate the advantages and disadvantages of OODBMSs. • Differentiate between Object-oriented data modeling & Conceptual data modeling. 10.2 Object-Oriented Database Design Before studying the object-oriented database design,
we start the discussion with a comparison of the basis for our methodology, the Enhanced Entity–Relationship model, and the main
object-oriented concepts.
We will also
examine the relationships that can exist between objects and how referential integrity can be handled.
The methodology for conceptual and logical database design is based on the Enhanced Entity–Relationship (EER) model and has similarities with OODM. A comparison between OODM and conceptual data modeling (CDM) is represented in table 10.1.

The main difference is the encapsulation of both state and behavior in an object, whereas CDM captures only state and has no knowledge of behavior. Thus, CDM has no concept of messages and consequently no provision for encapsulation. The similarity between the two approaches makes the conceptual and logical data modeling methodology a reasonable basis for a methodology for object-oriented
190 database design. Although this methodology is aimed primarily at relational database design, the model can be mapped with relative simplicity to the network and hierarchical models. The logical data model produced had many-to-many relationships and recursive relationships removed. These are unnecessary changes for object-oriented modeling and can be omitted, as they were introduced because
of the limited modeling power of the traditional data models. The use of normalization in the methodology is still important and should not be omitted for object-oriented database design. Normalization is used to improve the model so that it satisfies various constraints that avoid unnecessary duplication of data. The fact that we are dealing with objects does not mean that redundancy is acceptable.
Every attribute in an object is dependent on the object identity.
Table 10.1 Difference between
OODM and CDM. OODM CDM DIFFERENCE Object Entity Object includes behavior Attribute Attribute None Association Relationship Associations are the same but inheritance in OODM includes both state and behavior Message - No corresponding concept in CDM Class Entity type/Supertype None Instance Entity None
191 Encapsulation No corresponding concept in CDM
Object-oriented database design requires the database schema to include both a description of the object data structure and constraints, and the object behavior. Relationships are represented in an object-oriented data model using reference attributes, typically implemented using OIDs. In the methodology, we decomposed all non-binary relationships (for example, ternary relationships) into binary relationships.
Here, we will
discuss how to represent binary relationships based on their cardinality:
one-to-one (1:1), one-to-many (1:*), and many-to-many (*:*).
Relationships • 1:1
relationships: A 1:1 relationship between objects A and B is represented by adding a reference attribute to object A and, to maintain referential integrity, a reference attribute to object B. • 1:* relationships: A 1:* relationship between
objects A and B is represented by adding a reference attribute to object B and an attribute containing a set of references to object A. • *:*
relationships: A *:* relationship between objects A and B is represented by adding an attribute containing a set of references to each object.
For
relational database design, we would decompose the *:* relationship into two 1:* relationships linked by an intermediate entity.
The EER approach by itself is insufficient to complete the design of an object- oriented database. The EER approach must be supported with a technique that identifies and documents the behavior of each class of object. This involves a detailed analysis of the processing requirements of the enterprise. In a conventional data flow approach using Data Flow Diagrams (DFDs), for example, the processing requirements of the system are analyzed separately from the data
Behavioral Design
192
model. In object-oriented analysis, the processing requirements are mapped onto a set of methods that are unique for each class. The methods that are visible to the user
or to other objects (public methods) must be distinguished from methods that are purely internal to a class (private methods). We can identify three types of public and private methods: • Constructors and destructors: Constructor methods generate new instances of a class and each new instance is given a unique OID. Destructor methods delete class instances that are no longer required. In some systems, destruction is an automatic process: whenever an object becomes inaccessible from other objects, it is automatically deleted. • Access methods: Access methods return the value of an attribute or set of attributes of a class instance. It may return a single attribute value, multiple attribute values, or a collection of values.
An access method may also return data relating to the class.
An access method may also derive data from an attribute.
Some systems automatically generate a method to access each attribute. This is the approach taken in the SQL:2011 standard, which provides an automatic observer (get) method for each attribute of each new data type • Transform methods: Transform methods change (transform) the state of a class instance.

Some systems automatically generate a method to update each attribute. Again, this is the approach taken in the SQL:2011 standard, which provides an automatic mutator (put) method for each attribute of each new data type • Identifying methods: There are several methodologies for identifying methods, which typically combine the following approaches: ? identify the classes and determine the methods that may be usefully provided for each class;
193 ? decompose the application in a top-down fashion and determine the methods that are required to provide the required functionality. 10.3
Persistence in OODBMSs
DBMSs are primarily concerned with the creation and maintenance of large, long- lived collections of data. As we have already seen from earlier chapters, modern DBMSs are characterized by their support of the following features: • A data model: A particular way of describing data, relationships between data, and constraints on the data. • Data persistence: The ability for data to outlive the execution of a program and possibly the lifetime of the program itself. • Data sharing: The ability for multiple applications (or instances of the same one) to access common data, possibly at the same time. • Reliability: The assurance that the data in the database is protected from hardware and software failures. • Scalability: The ability to operate on large amounts of data in simple ways. • Security and integrity: The protection of the data against unauthorized access, and the assurance that the data conforms to specified correctness and consistency rules. • Distribution: The ability to physically distribute a logically interrelated collection of shared data over a computer network, preferably making the distribution transparent to the user. In contrast, traditional programming languages provide constructs for procedural control and for data and functional abstraction but lack built-in support for many of the mentioned database features. Although each is useful in its respective domain, there exists an increasing number of applications that require functionality from both DBMSs and programming languages. Such applications
194 are characterized by their need to store and retrieve large amounts of shared, structured data. Since 1980 there has been considerable effort expended in developing systems that integrate the concepts from these two domains. However, the two domains have slightly different perspectives that have to be considered and the differences addressed. Perhaps two of the most important concerns from the programmers' perspective are performance and ease of use, both achieved by having a more seamless integration between the programming language and the DBMS than that provided with traditional DBMSs. With a traditional DBMS, we find that it is the programmer's responsibility to decide when to read and update objects (records); the programmer has to write code to translate between the application's object model and the data model of the DBMS (for example, relations), which might be quite
different;
it is the programmer's responsibility to perform additional type- checking when an object is read back from the database. For example, the programmer may create an object in the strongly typed object-oriented language Java and store it in a traditional DBMS. However, another application written in a different language may modify the object, with no guarantee that the object will conform to its original type. These difficulties stem from the fact that conventional DBMSs have a two-level storage model: the application storage model in main or virtual memory, and the database storage model on disk, as illustrated in Figure 10.1 (
a).
In contrast, an OODBMS tries to give the illusion of a single-level storage model, with a similar representation in both memory and in the database stored on disk, as illustrated in Figure 10.1 (
b).
Although the single-level memory model looks intuitively simple, the OODBMS has to cleverly manage the representations of objects in memory and on disk to achieve this illusion. As we already know, objects, and relationships between objects, are identified by object identifiers (OIDs). There are two types of OID: •
logical OIDs that are independent of the physical location of the object on
195 disk. • physical OIDs that encode the location. In the former case, a level of indirection is required to look up the physical address of the object on the disk. In both cases, however, an OID is different in size from a standard in-memory pointer that
needs to
be large enough only to address all virtual memory. Thus, to achieve the required performance, an OODBMS must be able to convert OIDs to and from in-memory pointers. This conversion technique has become known as pointer swizzling or object faulting, and the approaches used to implement it have become varied, ranging from software-based residency checks to page faulting schemes used by the underlying hardware (Moss and Eliot, 1990). (
a)
196 (b) Figure 10.1 (a)
Two-level storage model for conventional (relational) DBMS. (b) Single-level storage model for OODBMS.
An object can be accessed on secondary storage that can have a significant impact on OODBMS performance,
in the following steps: •

The DBMS determines the page on secondary storage that contains the required record using indexes or table scans, as appropriate. The DBMS then reads that page from secondary storage and copies it into its cache. • The DBMS subsequently transfers the required parts of the record from the cache into the application's memory space. Conversions may be necessary to convert the SQL data types into the application's data types. • The application can then update the record's fields in its own memory space. • The application transfers the modified fields back to the DBMS cache using SQL, again requiring conversions between data types. • Finally, at an appropriate point, the DBMS writes the updated page of the cache back to secondary storage.
197
Figure 10.2 Steps in accessing a record using a conventional DBMS.
In contrast, with a single-level storage model, an OODBMS uses the following steps to retrieve an object from secondary storage, as illustrated in Figure 10.3: • The OODBMS determines the page on secondary storage that contains the required object using its OID or an index, as appropriate. The OODBMS then reads that page from secondary storage and copies it into the application's page cache within its memory space. • The OODBMS may then carry out a number of conversions, such as: ? swizzling references (pointers) between objects; ? adding some information to the object's data structure to make it conform to
the requirements of
the programming language;
198 ? modifying the data representations for data that has come from a different hardware platform or programming language. • The application can then directly access the object and update it, as required. • When the application wishes to make the changes persistent, or when the OODBMS needs to swap the page out of the page cache, the OODBMS may need to carry out similar conversions as listed earlier before copying the page back to secondary storage.
Figure 10.3 Steps in accessing an object using an OODBMS. 10.4
Encapsulation The concept of encapsulation is one of the main characteristics of Object-Oriented languages and systems. It is also related to the concepts of abstract data types and information hiding in programming languages. In traditional database models and systems, this concept was not applied, since it is customary to make the structure of database objects visible to users and external programs. In these traditional models, a number of standard database operations are applicable to
199 objects of all types. For example, in the relational model, the operations for selecting, inserting, deleting, and modifying tuples are generic and may be applied to any relation in the database. The relation and its attributes are visible to users and to external programs that access the relation by using these operations. Specifying Object Behavior via Class Operations The concepts of information hiding and encapsulation can be applied to database objects. The main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is accessible only through a number of predefined operations. Some operations may be used to create (insert) or destroy (delete) objects; other operations may update the object state; and others may be used to retrieve parts of the object state or to apply some calculations. Still other operations may perform a combination of retrieval, calculation, and update. In general, the implementation of an operation can be specified in a general-purpose programming language that provides flexibility and power in defining the operations. The external users of the object are only made aware of the interface of the object type, which defines the name and arguments (parameters) of each operation. The implementation is hidden from the external users; it includes the definition of the internal data structures of the object and the implementation of the operations that access these structures. In Object-Oriented terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. Typically, a method is invoked by sending a message to the object to execute the corresponding method. It should be noted that, as part of executing a method, a subsequent message to another object may be sent, and this mechanism may be used to return values from the objects to the external environment or to other objects. For database applications, the requirement that all objects be completely

200 encapsulated is too stringent. One way of relaxing this requirement is to divide the structure of an object into visible and hidden attributes (instance variables). Visible attributes may be directly accessed for reading by external operators, or by a high-level query language. The hidden attributes of an object are completely encapsulated and can be accessed only through predefined operations. Most OODBMSs employ high-level query languages for accessing visible attributes. The term class is often used to refer to an object type definition, along with the definitions of the operations for that type. A number of operations are declared for each class, and the signature (interface) of each operation is included in the class definition. A method (implementation) for each operation must be defined elsewhere, using a programming language. Typical operations include the object constructor operation, which is used to create a new object, and the destructor operation, which is used to destroy an object. A number of object modifier operations can also be declared to modify various attributes of an object. Additional operations can retrieve information about the object. Specifying Object Persistence via Naming and Reachability An OODBMS is often closely coupled with an OOPL. The OOPL is used to specify the method implementations as well as other application code. An object is typically created by some executing application program, by invoking the object constructor operation. Not all objects are meant to be stored permanently in the database. Transient objects exist in the executing program and disappear once the program terminates. Persistent objects are stored in the database and persist after program termination. The typical mechanisms for making an object persistent are naming and reachability. The naming mechanism involves giving an object a unique persistent name through which it can be retrieved by this and other programs. This persistent object name can be given via a specific statement or operation in the program. All such names given to objects must be unique within a particular database. Hence, the named persistent objects are used as entry points to the database through
201 which users and applications can start their database access. Obviously, it is not practical to give names to all objects in a large database that includes thousands of objects, so most objects are made persistent by using the second mechanism, called reachability. The reachability mechanism works by making the object reachable from some persistent object. An object B is said to be reachable from an object A if a sequence of references in the object graph lead from object A to object B. If we first create a named persistent object N, whose state is a set or list of objects of some class C, we can make objects of C persistent by adding them to the set or list, and thus making them reachable from N. Hence, N defines a persistent collection of objects of class C. 10.5 Complex Objects A principal motivation that led to the development of Object-Oriented systems was the desire to represent complex objects. There are two main types of complex objects: structured and unstructured. A structured complex object is made up of components and is defined by applying the available type constructors recursively at various levels. An unstructured complex object typically
is a data type that requires a large amount of storage, such as a
data type that represents an image or a large textual object. A structured complex object is defined by repeated application of the type constructors provided by the OODBMS. Hence, the object structure is defined and known to the OODBMS. As an example, consider the DEPARTMENT object. At the first level, the object has a tuple structure with six attributes: DNAME , DNUMBER , MGR , LOCATIONS , EMPLOYEES , and PROJECTS. However, only two of these attributes—namely, DNAME and DNUMBER —have basic values; the other four have complex values and hence build the second level of the complex object structure. One of these four (MGR) has a tuple structure, and the other three (LOCATIONS , EMPLOYEES , PROJECTS) have set structures. At the third
202 level, for a MGR tuple value, we have one basic attribute (MANAGERSTARTDATE) and one attribute (MANAGER) that refers to an employee object, which has a tuple structure. For a LOCATIONS set, we have a set of basic values, but for both the EMPLOYEES and the PROJECTS sets, we have sets of tuple-structured objects. Two types of reference semantics exist between a complex object and its components at each level. The first type, which we can call ownership semantics, applies when the sub-objects of a complex object are encapsulated within the complex object and are hence considered part of the complex object. The second type, which we can call reference semantics, applies when the components of the complex object are themselves independent objects but may be referenced from the complex object. The first type is also referred to as the is-part- of or is-component-of relationship; and the second type is called the is-associated- with relationship, since it describes an equal association between two independent objects. The is-part-of relationship (ownership semantics) for constructing complex objects has the property that the component objects are encapsulated within the complex object and are considered part of the internal object state. They need not have object identifiers and can only be accessed by methods of that object. They are deleted if the object itself is deleted. On the other hand, a complex object whose components are referenced is considered to consist of independent objects that can have their own identity and methods. When a complex object needs to access its referenced components, it must do so by invoking the appropriate methods of the components, since they are not encapsulated within the complex object. Hence, reference semantics represents relationships among independent objects. In addition, a referenced component object may be referenced by more than one complex object and hence is not automatically deleted when the complex object is deleted.

203 10.6 Issues in OODBMS Although OODBMSs are beneficial to deal with the complex objects, still there are some prominent issues that can affect the database system. Some major issues are discussed in this section. • Transactions
A transaction is a logical unit of work, which should always transform the database from one consistent state to another. The types of transactions found in business applications are typically of short duration. In contrast, transactions involving complex objects, such as those found in engineering and design applications, can continue for several hours, or even several days. Clearly, to support long-duration transactions we need to use different protocols from those used for traditional database applications, in which transactions are typically of a very short duration. In an OODBMS, the unit of concurrency control and recovery is logically an object, although for performance reasons a more coarse granularity may be used. Locking-based protocols are the most common type of concurrency control mechanism used by OODBMSs to prevent conflict from occurring. However, it would be totally unacceptable for a user who initiated a long-duration transaction to find that the transaction had been aborted owing to a lock conflict and the work had been lost. Two of the solutions that have been proposed are: Multiversion concurrency control protocols
and
Advanced transaction models, such as nested transactions, sagas, and multilevel
transactions. •
Versions There are many applications that need access to the previous state of an object. For example, the development of a particular design is often an experimental and incremental process, the scope of which changes with time. It is therefore necessary in databases that store designs to keep track of the evolution of design objects and the changes made to a design by various
204 transactions (for example, Atwood, 1985; Katz et al., 1986; Banerjee et al., 1987a). The process of maintaining the evolution of objects is known as version management. An object version represents an identifiable state of an object; a version history represents the evolution of an object. Versioning should allow changes to the properties of objects to be managed in such a way that object references always point to the correct version of an object. Figure 10.4 illustrates version management for three objects:
O A , O B , and O C . For example, we can determine that object O A
consists of versions V 1 , V 2 , V 3 ; V 1A is derived from V 1 , and V 2A and V 2B are derived from V 2 . This figure also shows an example of a configuration of objects, consisting of V 2B of
O A , V 2A of O B , and V 1B of O C .
Figure 10.4 Versions and Configurations •
Schema Evolution Design is an incremental process and evolves with time. To support this process, applications require considerable flexibility in dynamically defining and modifying the database schema. For example, it should be possible to modify class definitions, the inheritance structure, and the specifications of attributes and methods without requiring system shutdown. Schema modification is closely related to the concept of version management discussed
205 earlier. The issues that arise in schema evolution are complex and not all of them have been investigated in sufficient depth. Typical changes to the schema include (Banerjee et al., 1987b): 1. Changes to the class definition: - modifying attributes - modifying methods. 2. Changes to the inheritance hierarchy: - making a class S the superclass of a class C; - removing a class S from the list of superclasses of C; - modifying the order of the superclasses of C. 3. Changes to the set of classes, such as creating and deleting classes and modifying class names. The changes proposed to a schema must not leave the schema in an inconsistent state. Itasca and GemStone define rules for schema consistency, called schema invariants, which must be complied with as the schema is modified.
The rules can be divided into four groups with the following responsibilities: 1. The resolution of conflicts caused by multiple inheritance and the redefinition of attributes and methods in a subclass. a) Rule of precedence of subclasses over superclasses If an attribute/ method of one class is defined with the same name as an attribute/ method of a superclass, the definition specified in the subclass takes precedence over the definition of the superclass.
b) Rule
of precedence between superclasses of a different origin If several superclasses have attributes/methods with the same name but with a different origin, the attribute/method of the first superclass
is
inherited by the subclass.
206 c) Rule of precedence between superclasses of the same origin If several superclasses have attributes/ methods with the same name and the same origin, the attribute/ method is inherited only once. If the domain of the attribute has been redefined in any superclass, the attribute with the most specialized domain is inherited by the subclass. If domains cannot be compared, the attribute is inherited from the first superclass. 2.
The propagation of modifications to subclasses. a)
Rule for propagation of modifications Modifications to an attribute/ method in a class are always inherited by subclasses, except by those subclasses in which the attribute/ method has been redefined.

b)
Rule for propagation of modifications in the event of conflicts The introduction of a new attribute/ method or the modification of the name of an attribute/method is propagated only to subclasses for which there would be no resulting name conflict.
c)
Rule for modification of domains The domain of an attribute can be modified only using generalization. The domain of an inherited attribute cannot be made more general than the domain of the original attribute in the superclass. 3. The aggregation and deletion of inheritance relationships between classes and the creation and removal of classes.
a)
Rule for inserting superclasses If a class C is added to the list of superclasses of a class C s , C becomes the last of the superclasses of C s . Any resulting inheritance conflict is resolved by rules 1.
a), 1.b), and 1.c). b)
Rule for removing superclasses If a class C has a single superclass
C s , and C s
is deleted from the list of
207 superclasses of C, then C becomes a direct subclass of each direct superclass of
C s .
The ordering of the new superclasses of C is the same as that of the superclasses of
C s .
c)
Rule for inserting a class into a schema If C has no specified superclass, C becomes the subclass of OBJECT (the root of the entire schema). d) Rule for removing a class from a schema To delete a class C from a schema, rule 3.b) is applied successively to remove C from the list of superclasses of all its subclasses. OBJECT cannot be deleted. 4. Handling of composite objects. The fourth group relates to those data models that support the concept of composite objects. This group has one rule, which is based on different types of composite
objects (
Banerjee et al. (1987b) and Kim et al. (1989)). • Architecture Here, we discuss two architectural issues: how best to apply the client–server architecture to the OODBMS environment, and the storage of methods. Client-Server
architecture: a)
Object server: This approach attempts to distribute the processing between the two components. Typically, the server process is responsible for managing storage, locks, commits to secondary storage, logging and recovery, enforcing security and integrity, query optimization, and executing stored procedures. The client is responsible for transaction
Many commercial OODBMSs are based on the client–server architecture to provide data to users, applications, and tools in a distributed environment. However, not all systems use the same client–server model. We can distinguish three basic architectures for a client–server DBMS that vary in the functionality assigned to each component (Loomis, 1992), as depicted in Figure 10.5:
208
management and interfacing to the programming
language. This is the best architecture for cooperative, object-to-object processing in an open, distributed environment.
b)
Page server: In this approach, most of the database processing is performed by the client. The server is responsible for secondary storage and providing pages at the client's request.
c)
Database server: In this approach, most of the database processing is performed by the server. The client simply passes requests to the server, receives results, and passes them on to the application. This is the approach taken by many RDBMSs.
Figure 10.5 Client–server architectures: (a) object server; (b) page server; (c) database server
In each case, the server resides on the same machine as the physical database; the client may reside on the same or different machine. If the client needs access to databases distributed across multiple machines, then the client communicates with a server on each machine. There may also be a number of clients communicating with one server; for example, one client for each user or application. Storing and executing methods: There are two approaches to handling methods:
209 store the methods in external files, as shown in Figure 10.6 (a), and store the methods in the database, as shown in Figure 10.7 (b). The first approach is similar to
the

function libraries or APIs found in traditional DBMSs, in which an application program interacts with a DBMS by linking in functions supplied by the DBMS vendor. With the second approach, methods are stored in the database and are dynamically bound to the application at runtime. The second approach offers several benefits: ? It eliminates redundant code: Instead of placing a copy of a method that accesses a data element in every program that deals with that data, the method is stored only once in the database. ? It simplifies modifications: Changing a method requires changing it in one place only. All the programs automatically use the updated method. Depending on the
nature of the change, rebuilding, testing, and redistribution of programs may be eliminated. ?
Methods are more secure: Storing the methods in the database gives them all the benefits of security provided automatically by the OODBMS. ? Methods can be shared concurrently: Again, concurrent access is provided automatically by the OODBMS. This also prevents multiple users making different changes to a method simultaneously. ? Improved integrity: Storing the methods in the database means that integrity constraints can be enforced consistently by the OODBMS across all applications. 10.7
Advantages and Disadvantages of OODBMSs OODBMSs can provide appropriate solutions for many types of advanced database applications. However, there are also
some disadvantages. Advantages •
Enriched modeling capabilities • Extensibility
210 • Removal of impedance mismatch • More expressive query language • Support for schema evolution • Support for long-duration transactions • Applicability to advanced database applications • Improved performance Disadvantages • Lack of universal data model • Lack of experience • Lack of standards • Competition • Query optimization compromises encapsulation • Locking at object level may impact performance • Complexity • Lack of support for views •
Lack of support for security 10.8 Summary • OODM provides
encapsulation of both state and behavior in an object, whereas
Conceptual data model (
CDM) captures only state and has no knowledge of behavior. CDM has no concept of messages and consequently no provision for encapsulation. •
Relationships are represented in an object-oriented data model using reference attributes, typically implemented using OIDs.
211 •
Data persistence is
the ability for data to outlive the execution of a program and possibly the lifetime of the program itself. •
To achieve the required performance, an OODBMS must be able to convert OIDs to and from in-memory pointers. This conversion technique has become known as pointer swizzling or object faulting. • The
concepts of information hiding and encapsulation can be applied to database objects. The internal structure of the object is hidden, and the object is accessible only through a number of predefined operations. • In Object-Oriented terminology, the interface part of each operation is called the signature, and the operation implementation is called a method. • A principal motivation that led to the development of Object-Oriented systems was the desire to represent complex objects. There are two main types of complex objects: structured and unstructured. 10.9 Key Terms •
Pointer Swizzling: The action of converting object identifiers to main memory pointers and back again. The aim of pointer swizzling is to optimize access to objects. •
Encapsulation: Encapsulation in object oriented means an object contains both the data structures and the methods to manipulate the data structures. The data structures are internal to the object and are only accessed by other objects through the public methods. Encapsulation provides a form
of data independence. • Security and integrity: The protection of the data against unauthorized access, and the assurance that the data conforms to specified correctness and consistency rules. •
Structured Complex Object: A structured complex object is made up of
212 components and is defined by applying the available type constructors recursively at various levels. • Unstructured Complex Object: An unstructured complex object typically
is a data type that requires a large amount of storage, such as a
data type that represents an image or a large textual object. 10.10 Check Your Progress Short- Answer Type Q1)
Every attribute in an object is dependent on the object identity. (
True/ False?) Q2)
Which of the following is not a feature of OODBMS? a) Data Sharing b) Data Persistence c) Distribution d) All of the above Q3) A _____ complex object requires a large amount of storage. Q4) The main
aim of pointer swizzling is to optimize access to objects. (
True/ False?) Q5) In Object-Oriented terminology, the interface part of each operation is called the _____ and the operation implementation is called a _____. Long- Answer Type Q1)

Discuss why schema control may be a useful facility for some applications.
Q2) How does the single-level storage model affect data access? Q3) State the advantages and disadvantages of OODBMS. Q4) What are complex objects? Explain briefly. Q5) Write a short note on Encapsulation.
213 References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007.
214 MODULE- VI DISTRIBUTED DATABASE
215
Unit 11 – Distributed Database Structure 11.0 Introduction 11.1 Unit Objectives 11.2 Basics of Distributed Database 11.3 DDBMS architectures 11.4 Homogeneous and Heterogeneous databases 11.5 Distributed data storage 11.6 Advantages & Disadvantages of Data Distribution 11.7 Summary 11.8 Key Terms 11.9 Check Your Progress 11.0 Introduction
In previous chapters we have concentrated on centralized database systems, that is, systems with a single logical database located at one site under the control of a single DBMS. Now, we will learn about the Distributed Database Management System (DDBMS), which allows users to access not only the
data at their own site but also data stored at remote sites.
A distributed database system consists of loosely coupled sites that share no physical components. Furthermore, the database systems that run on each site may have a substantial degree of mutual independence. Each site may participate in the execution of transactions that access data at one site, or several sites. The main difference between centralized and distributed database systems is that, in the former, the data reside in one single location, whereas in the latter, the data reside in several locations. This distribution of data is the cause of many
216 difficulties in transaction processing and query processing. In this unit, we will address these difficulties. Distributing data across sites in an organization allows those data to reside where they are generated or most needed, but still to be accessible from other sites and from other departments. Keeping multiple copies of the database across different sites also allows large organizations to continue their database operations even when one site is affected by a natural disaster, such as flood, fire, or earthquake. Distributed database systems handle geographically or administratively distributed data spread across multiple database systems. This unit introduces the basic concept of distributed databases and architecture of distributed database systems. The types of distributed databases, homogeneous and heterogeneous databases are also discussed. 11.1 Unit Objectives After completing this unit, the reader will be able to: ● Justify the need for distributed databases. ● Illustrate the architecture for a distributed DBMS. ● Learn about the homogeneous and heterogeneous distributed databases. ● Discuss the advantages and disadvantages of data distribution. 11.2 Basics of Distributed Database In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with one another through various communication media, such as high-speed private networks or the Internet. They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.
217 The computers in a distributed system are referred to by a number of different names, such as sites or nodes, depending on the context in which they are mentioned. We mainly use the term site, to emphasize the physical distribution of these systems. The general structure of a distributed system appears in figure 11.1. Figure 11.1 A distributed system The distributed database (

including the newly started Internet-2 development.

100% **MATCHING BLOCK 166/202** SA 47F417BA15858476660.pdf (D123781188)

With advances in distributed processing and distributed computing that occurred in the operating systems arena, the database research community did considerable work to address the issues of data distribution, distributed query and transaction processing, distributed database metadata management, and other topics, and developed many research prototypes. However, a full-scale comprehensive DDBMS that 218 implements the functionality and techniques proposed in DDB research never emerged as a commercially viable product. Most major vendors redirected their efforts from developing a "pure" DDBMS product into developing systems based on client-server, or toward developing

active heterogeneous DBMSs.

97% **MATCHING BLOCK 167/202** SA 47F417BA15858476660.pdf (D123781188)

We can define a distributed database (DDB) as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user. A collection of files stored at different nodes of a network and the maintenance of

inter relationships

100% **MATCHING BLOCK 168/202** SA 47F417BA15858476660.pdf (D123781188)

among them via hyperlinks has become a common organization on the Internet, with files of Web pages. The common functions of database management, including uniform query processing and transaction processing, do not apply to this scenario yet.

100% **MATCHING BLOCK 177/202** SA 47F417BA15858476660.pdf (D123781188)

Distributed databases bring the advantages of distributed computing to the database management domain. A distributed computing system consists of a number of processing elements, not necessarily homogeneous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks. As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it efficiently in a coordinated manner. 11.3

DDBMS architectures A DDBMS is expected
to have the following functionality: • extended communication services to provide access to remote sites and allow the transfer of queries and data among the sites using a network; • extended system catalog to store data distribution details; • distributed query processing, including query optimization and remote data access;
219 • extended security control to maintain appropriate authorization/ access privileges to the distributed data; • extended concurrency control to maintain consistency of distributed and possibly replicated data; • extended recovery services to take account of failures of individual sites and the failures of communication links.
It is much more difficult to present an equivalent architecture
for a DDBMS in comparison to the centralized DBMS.
However,
it may be useful to present one possible reference architecture that addresses data distribution. The reference architecture shown in Figure 11.2 consists of the following schemas:
Reference Architecture for
a
DDBMS •
a set of global external schemas; • a global conceptual schema; • a fragmentation schema and allocation schema; • a set of schemas for each local DBMS. The edges in this figure represent mappings between the different schemas. Depending on which levels of transparency are supported, some levels may be missing from the architecture. ?

Global conceptual schema: The global conceptual schema is a logical description of the whole database, as if it were not distributed. This level corresponds to the conceptual level of the ANSI-SPARC architecture and contains definitions of entities, relationships, constraints, security, and integrity information. It provides physical data independence from the distributed environment. The global external schemas provide logical data independence. ?
Fragmentation and allocation schemas: The fragmentation schema is a description of how the data is to be logically partitioned. The allocation
220 schema is a description of where the data is to be located, taking account of any replication. ? Local schemas: Each local DBMS has its own set of schemas. The local conceptual and local internal schemas correspond to the equivalent levels of the ANSI-SPARC architecture. The local mapping schema maps fragments in the allocation schema into external objects in the local database. It is DBMS independent and is the basis for supporting heterogeneous DBMSs.
Figure 11.2 Reference architecture
for a
DDBMS
221 Independent of the reference architecture, we can identify a
component
architecture for a DDBMS consisting of four major components: Component Architecture for a DDBMS • local DBMS (LDBMS) component; • data communications (DC) component; • global system catalog (GSC); • distributed DBMS (DDBMS) component. The component architecture for a DDBMS is illustrated in
figure 11.3.
It has two exactly the same structures on both the sites.
For clarity, we have omitted Site 2 from the diagram, as it has the same structure as Site 1. ?
Local DBMS component: The LDBMS component is a standard DBMS, responsible for controlling the local data at each site that has a database. It has its own local system catalog that stores information about the data held at that site. In a homogeneous system, the LDBMS
component is the same product, replicated at each site. In a heterogeneous system, there would be at least two sites with different DBMS products and/or platforms. ? Data communications component: The DC component is the software that enables all sites to communicate with each other. The DC component contains information about the sites and the links. ? Global system catalog: The GSC has the same functionality as the system catalog of a centralized system. The GSC holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas. It can itself be managed as a distributed database and so it can be fragmented and distributed, fully replicated, or centralized, like any other relation, as we discuss shortly. A fully replicated GSC compromises site autonomy, as every modification to the GSC has to be communicated to all other sites. A centralized GSC also compromises site
222 autonomy and is vulnerable to failure of the central site. ?
Distributed DBMS component: The DDBMS component is the controlling unit of the entire system.
Figure 11.3 Components Architecture for a DDBMS 11.4
Homogeneous and Heterogeneous databases A DDBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product. In such a system, local sites
surrender a portion of their autonomy in terms of their right to change schemas or database-management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.
In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs. different sites may use different schemas, and
different database-management system software. The sites may not be aware of one another,
and
they may provide only limited facilities for cooperation in transaction processing. The
differences in schemas are often a major problem for query processing, while the
223 divergence in software becomes a hindrance for processing transactions that access multiple sites.

Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites. Heterogeneous systems usually result when individual sites have implemented their own databases and integration is considered at a later stage. In a heterogeneous system, translations are required to allow communication between different DBMSs. To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation. Data may be required from another site that may have: • different hardware; • different DBMS products; • different hardware and different DBMS products. If the hardware is different but the DBMS products are the same, the translation is straightforward, involving the change of codes and word lengths. If the DBMS products are different, the translation is complicated involving the mapping of data structures in one data model to the equivalent data structures in another data model. For example, relations in the relational data model are mapped to records and sets in the network model. It is also necessary to translate the query language used (for example, SQL SELECT statements are mapped to the network FIND and GET statements). If both the hardware and software are different, then both these types of translation are required. This makes the processing extremely complex. An additional complexity is the provision of a common conceptual schema, which is formed from the integration of individual local conceptual schemas.
The
224 integration of data models can be very difficult owing to the semantic heterogeneity. For example, attributes with the same name in two schemas may represent different things. Equally well, attributes with different names may model the same thing.
The typical solution used by some relational systems that are part of a heterogeneous DDBMS is to use gateways, which convert the language and model of each different DBMS into the language and model of the relational system. However, the gateway approach has some serious limitations. First, it may not support transaction management, even for a pair of systems; in other words, the gateway between two systems may be only a query translator. For example, a system may not coordinate concurrency control and recovery of transactions that involve updates to the pair of databases. Second, the gateway approach is concerned only with the problem of translating a query expressed in one language into an equivalent expression in another language. As such, generally it does not address the issues of homogenizing the structural and representational differences between different schemas. 11.5
Distributed Data Storage Consider a relation r that is to be stored in the database. There are two approaches to storing this relation in the distributed database: • Replication: The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The
alternative to replication is to store only one copy of relation r. •
Fragmentation: The system partitions the relation into several fragments, and stores each fragment at a different site.
Fragmentation and replication can be combined: A relation can be partitioned into several fragments and there may be several replicas of each fragment.
225
Data Replication If relation r is replicated, a copy of relation r is stored in two or more sites. In the most extreme case, we have full replication, in which a copy is stored in every site in the system. There are a number of advantages and disadvantages to replication. 1. Availability: If one of the sites containing relation r fails, then the relation r can be found in another site.
Thus, the system can continue to process queries involving r, despite the failure of one site. 2. Increased parallelism: In the case where the majority of accesses to the relation r result in only the reading of the relation, then several sites can process queries involving r in parallel.
The more replicas of r there are, the greater the chance that the needed data will be found in the site where the transaction is executing.
Hence, data replication minimizes movement of data between sites. 3.
Increased overhead on update: The system must ensure that all replicas of a relation r are consistent; otherwise, incorrect computations may result. Thus, whenever r is updated, the update must be propagated to all sites containing replicas.
The result is increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary to ensure that the balance in a particular account agrees in
all sites. In general, replication enhances the performance of read operations and increases the availability of data to read-only transactions.

However, update transactions incur greater overhead. Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems. We can simplify the management of replicas of relation r by choosing one of them as the primary copy of r. For example, in a banking system, an account can be associated with the site in which the account has been opened. Similarly, in an airline-reservation system, a flight can be associated with the site at which the flight originates.
226 Data
Fragmentation If relation r is fragmented, r is divided into a number of fragments r 1 , r 2 , . . . , r n . These fragments contain sufficient information to allow reconstruction of the original relation r. There are two different schemes for fragmenting a relation: horizontal fragmentation
and vertical fragmentation.
Horizontal fragmentation splits the relation by assigning each tuple of r to one or more fragments. Vertical fragmentation splits the relation
by decomposing the scheme R of relation r. In horizontal fragmentation, a relation r is partitioned into a number of subsets, r 1 , r 2 , . . . , r n . Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed. Horizontal fragmentation is usually used to keep tuples at the sites where they are used the most, to minimize data transfer. In general, a horizontal fragment can be defined as a selection on the global relation r. That is, we use a predicate P i to construct fragment r i : r i = σ Pi (r) We reconstruct the relation r by taking the union of all fragments; that is: r = r 1 ∪ r 2 ∪ · · · ∪ r n In our example, the fragments are disjoint. By changing the selection predicates used to construct the fragments, we can have a particular tuple of r appear in more than one of the r i . In its simplest form, vertical fragmentation is the same as decomposition. Vertical fragmentation of r(R) involves the definition of several subsets of attributes R 1 , R 2 , . . . , R n of the schema R so that: R = R 1 ∪ R 2 ∪ · · · ∪ R n Each fragment r i of r is defined by: r i = Π Ri (r)
227 The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join: r = r 1 ⋈ r 2 ⋈ r 3 ⋈ · · · ⋈ r n One way of ensuring that the relation r can be reconstructed is to include the primary-key attributes of R in each R i . More generally, any superkey can be used. It is often convenient to add a special attribute, called a tuple-id, to the schema R. The tuple-id value of a tuple is a unique value that distinguishes the tuple from all other tuples. The tuple-id attribute thus serves as a candidate key for the augmented schema, and is included in each R i . The physical or logical address for a tuple can be used as a tuple-id, since each tuple has a unique address. The two types of fragmentation can be applied to a single schema; for instance, the fragments obtained by horizontally fragmenting a relation can be further partitioned vertically. Fragments can also be replicated. In general, a fragment can be replicated, replicas of fragments can be fragmented further, and so on. Transparency The user of a distributed database system should not be required to know where the data are physically located nor how the data can be accessed at the specific local site. This characteristic, called data transparency, can take several forms: Fragmentation transparency: Users are not required to know how a relation has been fragmented. Replication transparency: Users view each data object as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed. Location transparency: Users are not required to know the physical location of the data. The distributed database system should be able to find any data as long as the data identifier is supplied by the user transaction. Data items, such as relations, fragments, and replicas, must have unique names.
228 This property is easy to ensure in a centralized database. In a distributed database, however, we must take care to ensure that two sites do not use the same name for distinct data items. One solution to this problem is to require all names to be registered in a central name server. The name server helps to ensure that the same name does not get used for different data items. We can also use the name server to locate a data item, given the name of the item. This approach, however, suffers from two major disadvantages. First, the name server may become a performance bottle-neck when data items are located by their names, resulting in poor performance. Second, if the name server crashes, it may not be possible for any site in the distributed system to continue to run. 11.6
Advantages & Disadvantages of Data Distribution The distribution of data and applications has potential advantages over traditional centralized database systems. Unfortunately, there are also disadvantages. In this section we review the advantages and disadvantages of the DDBMS. Advantages • Reflects organizational structure: Many organizations are naturally distributed over several locations. It is natural for databases used in such an application to be distributed over these locations. •
Improved shareability and local autonomy: The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can

access data stored at other sites. Data can be placed at the site close to the users who normally use that data. In this way, users have local control of the data and they can consequently establish and enforce local policies regarding the use of this data. A global DBA is responsible for the entire system. Generally, part of this responsibility is devolved to the local level, so that the local DBA can manage the local DBMS.
229 • Improved availability: In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS or a failure of a communication link making some sites inaccessible does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. If a single node fails, the system may be able to reroute the failed node's requests to another site. • Improved reliability: Because data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible. • Improved performance: As the data is located near the site of "greatest demand," and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remote centralized database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralized DBMS. • Economics: In the 1960s, computing power was calculated according to the square of the costs of the equipment: three times the cost would provide nine times the power. This was known as Grosch's Law. However, it is now generally accepted that it costs much less to create a system of smaller computers with the equivalent power of a single large computer. This makes it more cost-effective for corporate divisions and departments to obtain separate computers. It is also much more cost-effective to add workstations to a network than to update a mainframe system. The second potential cost saving occurs where databases are geographically remote and the applications require access to distributed data. In such cases, owing to the relative expense of data being transmitted across the network as opposed to the cost of local access, it may be much more economical to partition the application and perform the processing locally at each site. • Modular growth: In a distributed environment, it is much easier to handle
230 expansion. New sites can be added to the network without affecting the operations of other sites. This flexibility allows an organization to expand relatively easily. Increasing database size can usually be handled by adding processing and storage power to the network. In a centralized DBMS, growth may entail changes to both hardware (the procurement of a more powerful system) and software (the procurement of a more powerful or more configurable DBMS).
The transition to the new hardware/software could give rise to many difficulties. •
Integration: At the start of this section we noted that integration was a key advantage of the DBMS approach, not centralization. The integration of legacy systems is one particular example that demonstrates how some organizations are forced to rely on distributed data
processing
to allow their legacy systems to coexist with their more modern systems. At the same time, no one package can provide all the functionality that an organization requires nowadays. Thus, it is important for organizations to be able to integrate software components from different vendors to meet their specific requirements. • Remaining competitive: There are a number of relatively recent developments that rely heavily on distributed database technology such as e-business, computer-supported
collaborative work, and workflow management. Many enterprises have had to reorganize their businesses and use distributed database technology to remain competitive.
Disadvantages • Complexity: A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS. The fact that data can be replicated also adds an extra level of complexity to the distributed DBMS. If the software does not handle data replication adequately, there will be degradation in availability, reliability, and
231 performance compared with the centralized system, and the advantages we cited earlier will become disadvantages. • Cost: Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional labor costs to manage and maintain the local DBMSs and the underlying network. • Security: In a centralized system, access to the data can be easily controlled. However, in a distributed DBMS
not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure. In
the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made to make networks more secure. • Integrity control
is

more difficult: Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Enforcing integrity constraints generally requires access to a large amount of data that defines the constraint but that is not involved in the actual update operation itself. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive. • Lack of standards: Although distributed DBMSs depend on effective communication, we are only now starting to see the appearance of standard communication
and data access protocols. This lack of standards has significantly limited the potential of distributed DBMSs. There are also no tools or methodologies to help users convert a centralized DBMS into a distributed DBMS. • Lack of experience: General-purpose distributed DBMSs have not been widely
232 accepted, although many of the protocols and problems are well understood. •
Database design is
more complex: Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication. 11.7
Summary •
A distributed database is a logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network. The DDBMS is the software that transparently manages the distributed database. • A
DDBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product. In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs. •
There are several issues involved in storing a relation in the distributed database, including replication and fragmentation. It is essential that the system minimize the degree to which a user needs to be aware of how a relation is stored. •
The DDBMS should appear like a centralized DBMS by providing a series of transparencies. With distribution transparency, users should not know that the data has been fragmented/replicated. With transaction transparency, the consistency of the global database should be maintained when multiple users are accessing the database concurrently and when failures occur. With performance transparency, the system should be able to efficiently handle queries that reference data at more than
233 one site. With DBMS transparency, it should be possible to have different DBMSs in the system. 11.8
Key Terms • Homogeneous database system:
In a homogeneous system, all sites use the same DBMS product. •
Heterogeneous database system:
In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs. •
Global conceptual schema: The global conceptual schema is a logical description of the whole database, as if it were not distributed. •
Replication: The database system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. • Fragmentation: The
database
system partitions the relation into several fragments, and stores each fragment at a different site. 11.9
Check Your Progress Short- Answer Type Q1) A distributed database has following advantage over centralized database- a) Software complexity b) Modular growth c) Software cost d) None of the above Q2) In _____ partitioning some of the columns of a relation are at different sites. Q3) Each site/ node in a distributed system is subject to the same types of failures as in a centralized system. (True/ False?) Q4) In data replication each site must have the same storage category. (True/
234 False?) Q5) Storing a separate copy of the database in multiple locations is known as _____. Long- Answer Type Q1)
What is the difference between a homogeneous DDBMS
and a heterogeneous DDBMS? Under what circumstances would such systems generally arise?
Q2) Discuss the advantages and disadvantages of a DDBMS. Q3)
What are the differences between horizontal and vertical fragmentation schemes?
Q4) Write a short note on Data Replication. Q5) What is component architecture of DDBMS? Explain briefly. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •

Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011. • Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007.
235 Unit 12 – Other Concepts of Distributed Databases Structure 12.0 Introduction 12.1 Unit Objectives 12.2 Distributed transactions 12.3 Concurrency control & recovery in distributed databases 12.3.1 Distributed Concurrency Control 12.3.2 Distributed Recovery 12.4 Directory systems 12.5 Commit Protocols & Availability 12.6 Data Allocation and Fragmentation 12.7 Distributed database transparency 12.8 Summary 12.9 Key Terms 12.10 Check Your Progress 12.0 Introduction We have already learnt about the basic concepts of distributed DBMS. In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with one another through various communication media, such as high-speed private networks or the Internet. They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.
236 There are some other concepts of DDBMS that are considered while designing such systems. Like in centralized DBMS, the concurrency control and recovery schemes are also given for the distributed DBMS. Although there are certain changes in the properties of distributed DBMS. This unit focuses on the transaction schemes of DDBMS, concurrency control and recovery in DDBMS, and the properties of commit protocols and availability. 12.1 Unit Objectives After completing this unit, the reader will be able to: • Justify the concept of distributed transactions. • Illustrate the concurrency control & recovery in distributed databases. • Learn about the directory systems, commit protocols, and availability. • Discuss the data allocation & fragmentation of DDBMS in detail. • Gain knowledge about the distributed database transparency. 12.2

100% **MATCHING BLOCK 169/202** W

Distributed Transactions Access to the various data items in a distributed system is usually accomplished through transactions, which must preserve the ACID properties. There are two types of

transactions

100% **MATCHING BLOCK 170/202** W

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases.

97% **MATCHING BLOCK 171/202** W

Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site. The various transaction managers cooperate to execute global transactions. To understand how such a manager can be implemented, consider an abstract model of a transaction system, in which each site contains two subsystems: 237 • The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

It should be noted that

93% **MATCHING BLOCK 172/202** W

each such transaction may be either a local transaction (that is, a transaction that executes at only that site) or part of a global transaction (that is, a transaction that executes at several sites). • The transaction coordinator coordinates the execution of the various transactions (both local and global) initiated at that site. The overall system architecture appears in figure 12.1.

Figure 12.1 System architecture of Distributed Transaction

98% MATCHING BLOCK 173/202 W

The structure of a transaction manager is similar in many respects to the structure of a centralized system. Each transaction manager is responsible for: • Maintaining a log for recovery purposes. • Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site. As we shall see, we need to modify both the recovery and concurrency schemes to accommodate the distribution of transactions. The transaction coordinator subsystem is not needed in the centralized environment, since a transaction accesses data at only a single site. A transaction coordinator, as its name implies, 238 is responsible for coordinating the execution of all the transactions initiated at that site. For each such transaction, the coordinator is responsible for: • Starting the execution of the transaction. • Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution. • Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

A

100% MATCHING BLOCK 174/202 W

distributed system may suffer from the same types of failure that a centralized system does (for example, software errors, hardware errors, or disk crashes). There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are: • Failure of a site. • Loss of messages. • Failure of a communication link. • Network partition. The loss or corruption of messages is always a possibility in a distributed system. The system uses transmission-control protocols, such as TCP/ IP, to handle such errors.

100% MATCHING BLOCK 175/202 W

However, if two sites A and B are not directly connected, messages from one to the other must be routed through a sequence of communication links. If a communication link fails, messages that would have been transmitted across the link must be rerouted. In some cases, it is possible to find another route through the network, so that the messages are able to reach their destination. In other cases, a failure may result in there being no connection between some pairs of sites. A system is partitioned if it has been split into two (or more) subsystems, called partitions, that lack any connection between them. Note that, under this definition, a partition may consist of a single node. 239 12.3

Concurrency control & Recovery in distributed databases For concurrency control and recovery purposes, numerous problems arise in a distributed DBMS environment that are not encountered in a centralized DBMS environment. These include the following: • Dealing with multiple copies of the data items: The concurrency control method is responsible for maintaining consistency among these copies. The recovery method is responsible for making a copy consistent with other copies if the site on which the copy is stored fails and recovers later. • Failure of individual sites: The DDBMS should continue to operate with its running sites, if possible, when one or more individual sites fail. When a site recovers, its local database must be brought up to date with the rest of the sites before it rejoins the system. • Failure of communication links: The system must be able to deal with failure of one or more of the communication links that connect the sites. An extreme case of this problem is that network partitioning may occur. This breaks up the sites into two or more partitions, where the sites within each partition can communicate only with one another and not with sites in other partitions. • Distributed commit: Problems can arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process.
The two-phase commit protocol is often used to deal with this problem. • Distributed deadlock: Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account. Distributed concurrency control and recovery techniques must deal with these and other problems.

240 12.3.1 Distributed Concurrency Control We are already aware of the concurrency control schemes in the DBMS. They can be modified according to the distributed environment. The protocols of concurrency control requires updates to be done on all replicas of a data item. If any site containing a replica of a data item has failed, updates to the data item cannot be processed. A. Locking Protocols The various locking protocols can be used in a distributed environment. The only change that needs to be incorporated is in the way the lock manager deals with replicated data. We present several possible schemes that are applicable to an environment where data can be replicated in several sites. 1. Single Lock-Manager Approach: In the single lock-manager approach, the system maintains a single lock manager that resides in a single chosen site, say S i . All lock and unlock requests are made at site S i . When a transaction needs to lock a data item, it sends a lock request to S i . The lock manager determines whether the lock can be granted immediately. If the lock can be granted, the lock manager sends a message to that effect to the site at which the lock request was initiated. Otherwise, the request is delayed until it can be granted, at which time a message is sent to the site at which the lock request was initiated. The transaction can read the data item from any one of the sites at which a replica of the data item resides. In the case of a write, all the sites where a replica of the data item resides must be involved in the writing. This scheme has advantages of simple implementation and deadlock handling. But the issue of Bottleneck appears as the site S i becomes a bottleneck, since all requests must be processed there. If the site S i fails, the concurrency controller is lost. Either processing must stop, or a recovery scheme must be used so that a backup site can take over lock management from S i .
241 2. Distributed Lock Manager: A compromise between the advantages and disadvantages can be achieved through the distributed-lock-manager approach, in which the lock-manager function is distributed over several sites. Each site maintains a local lock manager whose function is to administer the lock and unlock requests for those data items that are stored in
that site. When a transaction wishes to lock a data item Q that is
not replicated
and resides at site S i , a message is sent to the lock manager at site S i requesting a lock (in a particular lock mode). If data item Q is locked in an incompatible mode, then the request is delayed until it can be granted. Once it has determined that the lock request can be granted, the lock manager sends a message back to the initiator indicating that it has granted the lock request. The distributed-lock-manager scheme has the advantage of simple implementation, and reduces the degree to which the coordinator is a bottleneck. It has a reasonably low overhead, requiring two message transfers for handling lock requests, and one message transfer for handling unlock requests. However, deadlock handling is more complex, since the lock and unlock requests are no longer made at a single site. There may be intersite deadlocks even when there is no deadlock within a single site. 3. Primary Copy: When a system uses data replication, we can choose one of the replicas as the primary copy. For each data item Q, the primary copy of Q must reside in precisely one site, which we call the primary site of Q. When a transaction needs to lock a data item Q, it requests a lock at the primary site of Q. As before, the response to the request is delayed until it can be granted. The primary copy enables concurrency control for replicated data to be handled like that for unreplicated data. This similarity allows for a simple implementation. However, if the primary site of Q fails, Q is
242 inaccessible, even though other sites containing a replica may be accessible. 4. Majority Protocol: The majority protocol works this way: If data item Q is replicated in n different sites, then a lock-request message must be sent to more than one- half of the n sites in which Q is stored. Each lock manager determines whether the lock can be granted immediately (as far as it is concerned). As before, the response is delayed until the request can be granted. The transaction does not operate on Q until it has successfully obtained a lock on a majority of the replicas of Q. 5. Biased Protocol: The biased protocol is another approach to handling replication. The difference from the majority protocol is that requests for shared locks are given more favorable treatment than requests for exclusive locks. ● Shared locks: ● When a transaction needs to lock data item Q, it simply requests a lock on Q from the lock manager at one site that contains a replica of Q. Exclusive locks: B. Timestamping When a transaction needs to lock data item Q, it requests a lock on Q from the lock manager at all sites that contain a replica of Q. We have already discussed the principal idea behind the timestamping scheme in which each transaction is given a unique timestamp that the system uses in deciding the serialization order. Our first task, then, in generalizing the centralized scheme to a distributed scheme is to develop a scheme for generating unique timestamps. Then, the various protocols can operate directly to the non- replicated environment.
243 There are two primary methods for generating unique timestamps, one centralized and one distributed. In the centralized scheme, a single site distributes the timestamps. The site can use a logical counter or its own local clock for this purpose. In the distributed scheme, each site generates a unique local timestamp by using either a logical counter or the local clock. We obtain the unique global timestamp by concatenating the unique local timestamp with the site identifier, which also must be unique (figure 12.2). We use
the site identifier in the least significant position to ensure that

the global timestamps generated in one site are not always greater than those generated in another site. Figure 12.2 Generation of Unique Timestamps C. Deadlock Handling The deadlock-prevention and deadlock-detection algorithms can be used in a distributed system, provided that modifications are made. For example, we can use the tree protocol by defining a global tree among the system data items. Similarly, the timestamp-ordering approach could be directly applied to a distributed environment. Deadlock prevention may result in unnecessary waiting and rollback. Furthermore, certain deadlock-prevention techniques may require more sites to be involved in the execution of a transaction than would otherwise be the case. If

244 we allow deadlocks to occur and rely on deadlock detection, the main problem in a distributed system is deciding how to maintain the wait-for graph. Common techniques for dealing with this issue require that each site keep a local wait-for graph. The nodes of the graph correspond to all the transactions (local as well as non-local) that are currently either holding or requesting any of the items local to that site. In the centralized deadlock detection approach, the system constructs and maintains a global wait-for graph (the union of all the local graphs) in a single site: the deadlock-detection coordinator. Since there is communication delay in the system, we must distinguish between two types of wait-for graphs. The real graph describes the real but unknown state of the system at any instance in time, as would be seen by an omniscient observer. The constructed graph is an approximation generated by the controller during the execution of the controller's algorithm. Obviously, the controller must generate the constructed graph in such a way that, whenever the detection algorithm is invoked, the reported results are correct. Correct means in this case that, if a deadlock exists, it is reported promptly, and if the system reports a deadlock, it is indeed in a deadlock state. The global wait-for graph can be reconstructed or updated under these conditions: ● Whenever a new edge is inserted in or removed from one of the local wait-for graphs. ● Periodically, when a number of changes have occurred in a local wait-for graph. ● Whenever the coordinator needs to invoke the cycle-detection algorithm. When the coordinator invokes the deadlock-detection algorithm, it searches its global graph. If it finds a cycle, it selects a victim to be rolled back. The coordinator must notify all the sites that a particular transaction has been selected as victim. The sites, in turn, roll back the victim transaction.

245 Deadlock detection can be done in a distributed manner, with several sites taking on parts of the task, instead of it being done at a single site. However, such algorithms are more complicated and more expensive. 12.3.2 Distributed Recovery The recovery process in distributed databases is quite involved. We give only a very brief idea of some of the issues here. In some cases it is quite difficult even to determine whether a site is down without exchanging numerous messages with other sites. For example, suppose that site X sends a message to site Y and expects a response from Y but does not receive it. There are several possible explanations: ● The message was not delivered to Y because of communication failure. ● Site Y is down and could not respond. ● Site Y is running and sent a response, but the response was not delivered. Without additional information or the sending of additional messages, it is difficult to determine what actually happened. Another problem with distributed recovery is distributed commit. When a transaction is updating data at several sites, it cannot commit until it is sure that the effect of the transaction on every site cannot be lost. This means that every site must first have recorded the local effects of the transactions permanently in the local site log on disk. The two-phase commit protocol is often used to ensure the correctness of distributed commits. 12.4

95% MATCHING BLOCK 176/202 W

Directory systems Consider an organization that wishes to make data about its employees available to a variety of people in the organization; examples of the kinds of data include name, designation, employee-id, address, email address, phone number, fax 246 number, and so on. In the pre-computerization days, organizations would create physical directories of employees and distribute them across the organization. Even today, telephone companies create physical directories of customers. In general, a directory is a listing of information about some class of objects such as persons. Directories can be used to find information about a specific object, or in the reverse direction to find objects that meet a certain requirement. In the world of physical telephone directories, directories that satisfy lookups in the forward direction are called white pages, while directories that satisfy lookups in the reverse direction are called yellow pages. In today's networked world, the need for directories is still present and, if anything, even more important. However, directories today need to be available over a computer network, rather than in a physical (paper) form. Directory Access Protocols Directory information can be made available through Web interfaces, as many organizations, and phone companies in particular, do. Such interfaces are good for humans. However, programs too need to access directory information. Directories can be used for storing other types of information, much like file system directories. For instance, Web browsers can store personal bookmarks and other browser settings in a directory system. A user can thus access the same settings from multiple locations, such as at home and at work, without having to share a file system. Several directory access protocols have been developed to provide a standardized way of accessing data in a directory. The most widely used among them today is the Lightweight Directory Access Protocol (LDAP).

98% MATCHING BLOCK 178/202 W

In general a directory system is implemented as one or more servers, which service multiple clients. Clients use the application programmer interface defined by the directory system to communicate with the directory servers. Directory access protocols also define a data model and access control. 247 The X.500 directory access protocol, defined by the International Organization for Standardization (ISO), is a standard for accessing directory information. However, the protocol is rather complex, and is not widely used. The Lightweight Directory Access Protocol (LDAP) provides many of the X.500 features, but with less complexity, and is widely used. In the rest of this section, we shall outline the data model and access protocol details of LDAP . 12.5

Commit Protocols & Availability If we are to ensure atomicity, all the sites in which a transaction T executed must agree on the final outcome of the execution. T must either commit at all sites, or it must abort at all sites. To ensure this property, the transaction coordinator of T must execute a commit protocol. Among the simplest and most widely used commit protocols is the two-phase commit protocol (2PC). An alternative is the three-phase commit protocol (3PC), which avoids certain disadvantages of the 2PC protocol but adds to complexity and overhead. Two-Phase Commit We first describe how the two-phase commit protocol (2PC) operates during normal operation, then describe how it handles failures and finally how it carries out recovery and concurrency control. Consider a transaction T initiated at site S i , where the transaction coordinator is C i . When T completes its execution, that is, when all the sites at which T has executed inform C i that T has completed, then C i starts the 2PC protocol. ● Phase 1: C i adds the record >prepare T< to the log, and forces the log onto stable storage. It then sends a prepared T message to all sites at which T executed. On receiving such a message, the transaction manager at that site determines whether it is willing to commit its portion of T. If the answer is no, it adds a record >no T< to the log, and then responds by sending an

248 abort T message to C i . If the answer is yes, it adds a record >ready T< to the log, and forces the log (with all the log records corresponding to T) onto stable storage. The transaction manager then replies with a ready T message to C i . ● Phase 2: When C i receives responses to the prepare T message from all the sites, or when a prespecified interval of time has elapsed since the prepare T message was sent out, C i can determine whether the transaction T can be committed or aborted. Transaction T can be committed if C i received a ready T message from all the participating sites. Otherwise, transaction T must be aborted. Depending on the verdict, either a record >commit T< or a record >abort T< is added to the log and the log is forced onto stable storage. At this point, the fate of the transaction has been sealed. Following this point, the coordinator sends either a commit T or an abort T message to all participating sites. When a site receives that message, it records the message in the log. A site at which T executed can unconditionally abort T at any time before it sends the message ready T to the coordinator. Once the message is sent, the transaction is said to be in the ready state at the site. The ready T message is, in effect, a promise by a site to follow the coordinator's order to commit T or to abort T. To make such a promise, the needed information must first be stored in stable storage. Otherwise, if the site crashes after sending ready T, it may be unable to make good on its promise. Further, locks acquired by the transaction must continue to be held until the transaction completes. Since unanimity is required to commit a transaction, the fate of T is sealed as soon as at least one site responds abort T. Since the coordinator site S i is one of the sites at which T executed, the coordinator can decide unilaterally to abort T. The final verdict regarding T is determined at the time that the coordinator writes that verdict (commit or abort) to the log and forces that verdict to stable storage. In some implementations of the 2PC protocol, a site sends an acknowledge T message to the coordinator at the end of the second phase of the protocol. When
249 the coordinator receives the acknowledge T message from all the sites, it adds the record >complete T< to the log. Three-Phase Commit The three-phase commit (3PC) protocol is an extension of the two-phase commit protocol that avoids the blocking problem under certain assumptions. In particular, it is assumed that no network partition occurs, and not more than k sites fail, where k is some predetermined number. Under these assumptions, the protocol avoids blocking by introducing an extra third phase where multiple sites are involved in the decision to commit. Instead of directly noting the commit decision in its persistent storage, the coordinator first ensures that at least k other sites know that it intended to commit the transaction. If the coordinator fails, the remaining sites first select a new coordinator. This new coordinator checks the status of the protocol from the remaining sites; if the coordinator had decided to commit, at least one of the other k sites that it informed will be up and will ensure that the commit decision is respected. The new coordinator restarts the third phase of the protocol if some site knew that the old coordinator intended to commit the transaction. Otherwise the new coordinator aborts the transaction.

97% **MATCHING BLOCK 179/202** **W**

One of the goals in using distributed databases is high availability; that is, the database must function almost all the time. In particular, since failures are more likely in large distributed systems, a distributed database must continue functioning even when there are various types of failures. The ability to continue functioning even during failures is referred to as robustness. For a distributed system to be robust, it must detect failures, reconfigure the system so that computation may continue, and recover when a processor or a link is repaired. 12.6

Data Allocation and Fragmentation Both data allocation and data fragmentation are the factors considered while designing the distributed DBMS along with data replication. We are already aware
250 of the data replication concept. Here, we will discuss data allocation & fragmentation.
Data Allocation There are four alternative strategies regarding the placement of data: centralized, fragmented, complete replication, and selective replication. We now compare these strategies using the objectives identified earlier. 1. Centralized This strategy consists of a single database and DBMS stored at one site with users
distributed
across the network (we referred to this previously as distributed processing). Locality of reference is at its lowest as all sites, except the central site, have to use the network for all data accesses. This also means that communication costs are high. Reliability and availability are low, as a failure of the central site results in the loss of the entire database system. 2. Fragmented (or partitioned) This strategy partitions the database into disjoint fragments, with each fragment assigned to one site. If data items are located at the site where they are used most frequently, locality of reference is high. As there is no replication, storage costs are low; similarly, reliability and availability are low, although they are higher than in the centralized case, as the failure of a site results in the loss of only that site's data. Performance should be good and communications costs low if the distribution is designed properly. 3.

Complete replication This strategy consists of maintaining a complete copy of the database at each site. Therefore, locality of reference, reliability and availability, and performance are maximized. However, storage costs and communication costs for updates are the most expensive. To overcome some of these problems, snapshots are sometimes used. A snapshot is a copy of the data
251 at a given time. The copies are updated periodically so they may not be always up to date. Snapshots are also sometimes used to implement views in a distributed database to improve the time it takes to perform a database operation on a view. 4. Selective replication This strategy is a combination of fragmentation, replication, and centralization. Some data items are fragmented to achieve high locality of reference, and others that are used at many sites and are not frequently updated are replicated; otherwise, the data items are centralized. The objective of this strategy is to have all the advantages of the other approaches but none of the disadvantages. This is the most commonly used strategy, because of its flexibility.
Table 12.1 Comparison of strategies for data allocation Locality of Reference Reliability & Availability Performance Storage Costs Communication Costs Centralized Lowest
Lowest Unsatisfactor y Lowest Highest Fragmented High Low for item; high for system Satisfactory Lowest Low Complete Replication Highest Highest Best
of
read Highest High for update; Low for read Selective Replication High Low for item; high for system Satisfactory Average Low
252 Data Fragmentation Following are the
four reasons for fragmenting a relation: 1. Usage: In general, applications work with views rather than entire relations. Therefore, for data distribution, it seems appropriate to work with subsets of relations as the unit of distribution. 2. Efficiency: Data is stored close to where it is most frequently used. In addition, data that is not needed by local applications is not stored. 3. Parallelism: With fragments as the unit of distribution, a transaction can be divided into several subqueries that operate on fragments. This should increase the degree of concurrency, or parallelism, in the system, thereby allowing transactions that can do so safely to execute in parallel. 4. Security: Data not required by local applications is not stored and consequently not available to unauthorized users.
Fragmentation
cannot be carried out haphazardly. There are three rules that must be followed during fragmentation:
a)
Completeness: If a relation instance R is decomposed into fragments R 1 , R 2 , . . . , R n , each data item that can be found in R must appear in at least one fragment. This rule is necessary to ensure that there is no loss of data during fragmentation. b) Reconstruction: It must be possible to define a relational operation that will reconstruct the relation R from the fragments. This rule ensures that functional dependencies are preserved.
c) Disjointness: If a data item d i , appears in fragment R i ,
then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction. This rule ensures minimal data redundancy. In the case of horizontal fragmentation, a data item is a tuple; for vertical fragmentation, a data item is an attribute.
253 Figure 12.3 (a) Horizontal and (b) vertical
fragmentation.
There are two main types of fragmentation: horizontal and vertical. Horizontal fragments are subsets of tuples and vertical fragments are subsets of attributes, as illustrated in figure 12.3.
Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions. A horizontal fragment is produced by specifying a predicate that performs a restriction on the tuples in the relation. It is defined using the Selection operation of the relational algebra. The Selection operation groups together tuples that have some common property; for example, the tuples are all used by the same application or at the same site. Given a relation R, a horizontal fragment is defined as:
σ p (
R) where p is a predicate based on one or more attributes of the relation.
Sometimes, the choice of horizontal fragmentation strategy is obvious. However, in other cases, it is necessary to analyze the applications in detail. The analysis involves an examination of the predicates (or search conditions) used by transactions or queries in the applications. The predicates may be simple, involving single attributes, or complex, involving multiple attributes. The predicates for each attribute may be single-valued or multivalued. In the latter case, the values may be discrete or involve ranges of values.

254 The fragmentation strategy involves finding a set of minimal (that is, complete and relevant) predicates that can be used as the basis for the fragmentation schema. A set of predicates is complete if and only if any two tuples in the same fragment are referenced with the same probability by any transaction. A predicate is relevant if there is at least one transaction that accesses the resulting fragments differently.
Vertical fragmentation groups together the attributes in a relation that are used jointly by the important transactions. A vertical fragment is defined using the Projection operation of the relational algebra. Given a relation R, a vertical fragment is defined as:
Па 1 , ..., a n (R) where a 1 , ... , a n are attributes of the relation R .
Vertical
fragments are determined by establishing the affinity of one attribute to another. One way to do this is to create a matrix that shows the number of accesses that refer to each attribute pair. 12.7
Distributed Database Transparency
The definition of a DDBMS states that the system should make the distribution transparent to the user. Transparency hides implementation details from
the user. For example, in a centralized DBMS data independence is a form of transparency, it hides changes in the definition
and organization of the data from the user. A DDBMS may provide various levels of transparency. However, they all participate in the same overall objective: to make the use of the distributed database equivalent to that of a centralized database. We can identify four main types of transparency
in a DDBMS: ● Distribution transparency ● Transaction transparency ● Performance transparency
255 ● DBMS transparency
Before we
discuss each of these transparencies, it is worthwhile noting that full
transparency
is not a universally accepted objective. Full transparency makes the management of distributed data very difficult and that applications coded with transparent access to geographically distributed databases have poor manageability, poor modularity, and poor message performance. 1.
Distribution Transparency
Distribution transparency allows the user to perceive the database
as a single, logical entity. If a DDBMS exhibits distribution transparency, then the user does not need to know the data is fragmented (fragmentation transparency)
or the location of data items (
location transparency). If the user needs to know that the data is fragmented and the location of fragments, then we call this
local mapping transparency.
a)
Fragmentation transparency: Fragmentation is the highest level of distribution transparency. If fragmentation transparency
is provided by the DDBMS,
then
the user does not need to know that the data
is fragmented.
As a result, database accesses are based on the global schema, so the user does not need to specify fragment names or data locations.
b)
Location
transparency: Location is the middle level of distribution transparency. With location transparency, the user must know how the data has been fragmented but still does not have to know the location of the data.
The
main advantage of location transparency is
that the database may be physically reorganized without impacting on the application programs that access them. c)
Replication transparency: Closely related to location transparency is replication transparency, which means that the user is unaware of the replication of fragments. Replication transparency is implied by location transparency.
However, it is possible for a system not to have location transparency but to have replication transparency.
256
d)

Local mapping transparency: This is the lowest level of distribution transparency. With local mapping transparency, the
user needs to specify both fragment names and the location of data items,
taking into consideration any replication that may
exist.
e)
Naming transparency: As a corollary to the previous distribution transparencies, we have naming transparency. As in a centralized database, each item in a distributed database must have a unique name. Therefore, the
DDBMS must ensure that no two sites create a database object with the same name.
One solution to this problem is to create a central name server, which has the responsibility for ensuring uniqueness of all names in the system. However, this approach results in: ● loss of some local autonomy; ● performance problems, if the central site becomes a bottleneck; ● low availability; if the central site fails, the remaining sites cannot create any new database objects. 2.
Transaction Transparency Transaction transparency in a DDBMS environment ensures that all distributed transactions maintain the distributed database's integrity and consistency. A distributed transaction accesses data stored at more than one location. Each transaction is divided into a number of subtransactions, one for each site that has to be accessed; a subtransaction is represented by an agent.
The
atomicity of the distributed transaction is still fundamental to the transaction concept, but in addition the DDBMS must also ensure the atomicity of each subtransaction. Therefore, not only must the DDBMS ensure synchronization of subtransactions with other local transactions that are executing concurrently at a site, but it must also ensure synchronization of subtransactions with global transactions running simultaneously at the same or different sites.
Transaction transparency in a distributed DBMS is complicated by the fragmentation, allocation, and replication schemas.
We consider two further aspects of
257 transaction transparency: concurrency transparency and failure transparency.
a)
Concurrency transparency Concurrency transparency is provided by the DDBMS if the results of all concurrent transactions (distributed and non-distributed) execute independently and are logically consistent with the results that are obtained
if the transactions are executed one at a time, in
some arbitrary serial order. These are the same fundamental principles for the centralized DBMS. However, there is the added complexity that the DDBMS must ensure that both global and local transactions do not interfere with each other. Similarly, the DDBMS must ensure the consistency of all subtransactions of the global transaction. Replication makes the issue of concurrency more complex. If a copy of a replicated data item is updated, the update must eventually be propagated to all copies. An obvious strategy is to propagate the changes as part of the original transaction, making it an atomic operation. However, if one of the sites holding a copy is not reachable when the update is being processed, because either the site or the communication link has failed, then the transaction is delayed until the site is reachable. If there are many copies of the data item, the probability of the transaction succeeding decreases exponentially. An alternative strategy is to limit the update propagation to sites that are currently available. The remaining sites must be updated when they become available again. A further strategy would be to allow the updates to the copies to happen asynchronously, sometime after the original update. The delay in regaining consistency may range from a few seconds to several hours.
b) Failure Transparency
A centralized DBMS must provide a recovery mechanism that ensures that, in the presence of failures, transactions are atomic: either all the operations
258 of the transaction are carried out or none at all. Furthermore, once a transaction has committed the changes are durable. We also examined the types of failure that could occur in a centralized system, such as system crashes, media failures, software errors, carelessness, natural physical disasters, and sabotage. In the distributed environment, the DDBMS must also cater for: ● the loss of a message ● the failure of a communication link ● the failure of a site ● network partitioning 3.
Performance Transparency Performance transparency requires a DDBMS to perform as if it were a centralized DBMS. In a distributed environment, the system should
not suffer any performance degradation due to the distributed architecture, such as

the presence of the network. Performance transparency also requires the DDBMS to determine the most cost-effective strategy to execute a request. In a centralized DBMS, the query processor (QP) must evaluate every data request and find an optimal execution strategy, consisting of an ordered sequence of operations on the database. In a distributed environment, the
distributed query processor (DQP) maps a data request into an ordered sequence of operations on the local databases. It has the added complexity of taking into account the fragmentation, replication, and allocation schemas. The DQP has to decide which fragment to access; which copy of
a
fragment to use, if the fragment is replicated; which location to use. The DQP produces an execution strategy that is optimized with respect to some cost function. Typically, the costs associated with a distributed request include: the access time (I/O) cost involved in accessing the physical data on disk; the CPU time cost incurred when performing operations on data in main memory; the
259 communication cost associated with the transmission of data across the network. The first two factors are the only ones considered in a centralized system. In a distributed environment, the DDBMS must take account of the communication cost, which may be the most dominant factor in WANs with a bandwidth of a few kilobytes per second. In such cases, optimization may ignore I/O and CPU costs. However, LANs have a bandwidth comparable to that of disks, so in such cases optimization should not ignore I/O and CPU costs entirely.
One approach to query optimization minimizes the total cost of time that will be incurred in executing the query (Sacco and Yao, 1982). An alternative approach minimizes the response time of the query, in which case the DQP attempts to maximize the parallel execution of operations (Epstein et al., 1978). 4.
DBMS Transparency
DBMS transparency hides the knowledge that the local DBMSs may be different and is therefore applicable only to heterogeneous DDBMSs. It is one of the most difficult transparencies to provide as a generalization. 12.8
Summary ● There are two types of transactions

100% **MATCHING BLOCK 180/202** W

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases. ● The

100% **MATCHING BLOCK 181/202** W

The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

The transaction coordinator coordinates the execution of the various transactions (both local and global) initiated at that site. ●
The
definition and allocation of fragments are carried out strategically to achieve locality of reference, improved reliability and availability, acceptable performance, balanced storage capacities and costs, and minimal
260 communication costs. The three correctness rules of fragmentation are completeness, reconstruction, and disjointness. ● There are four allocation strategies regarding the placement of data: centralized (a single centralized database), fragmented (fragments assigned to one site), complete replication (complete copy of the database maintained at each site), and selective replication (combination of the first three). ● The DDBMS should appear like a centralized DBMS by providing a series of transparencies. With distribution transparency, users should not know that the data has been fragmented/ replicated. With transaction transparency, the consistency of the global database should be maintained when multiple users are accessing the database concurrently and when failures occur. With performance transparency, the system should be able to efficiently handle queries that reference data at more than one site. With DBMS transparency, it should be possible to have different DBMSs in the system. 12.9
Key Terms ● Robustness:

90% **MATCHING BLOCK 182/202** W

The ability to continue functioning even during failures is referred to as robustness. ● Snapshot: A

snapshot is a copy of the data at a given time. ●
Fragmentation:
This strategy partitions the database into disjoint fragments, with each fragment assigned to one site. ●
Distribution Transparency: With
distribution transparency,
users should not know that the data has been fragmented/ replicated. ●
Performance Transparency:
With performance transparency, the system should be able to efficiently handle queries that reference data at more than one site.
261 12.10
Check Your Progress Short- Answer Type Q1) Commit and rollback are related to _____. Q2) Global Wait-for graph is used for Deadlock Handling in Distributed databases. (True/ False?) Q3) Which of the following commit protocols can avoid Blocking problem? a) Two-phase commit protocol b) Three-phase commit protocol c) Both of the above d) None of the above Q4)
All sites in a distributed database commit at exactly the same instant. (True/ False?)
Q5) _____ is storing a separate copy of the database at multiple locations. Long- Answer Type Q1) What are distributed transactions? Explain with the help of an example. Q2) How concurrency control schemes are different for distributed databases from that of centralized databases? Q3) Write a short note on Directory systems in DDBMS. Q4) What are data allocation and fragmentation? Give details. Q5) Discuss the types of distributed database transparency. Explain them briefly. References ●
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. ●
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
262
MODULE VII OBJECT RELATIONAL & EXTENDED RELATIONAL DATABASES
263
Unit: 13 Introduction to Relational Databases Structure 13.0 Introduction 13.1 Unit Objectives 13.2 Basic Concept of Relational Databases 13.3 Relational Database Design 13.4 Integrity Constraints 13.5 Introduction to ORDBMS 13.6 Summary 13.7 Key Terms 13.8 Check Your Progress 13.0 Introduction In the preceding units we have primarily discussed three data models, the Entity- Relationship (ER) model, the relational data model, and the object-oriented data model. We discussed how all these data models have been thoroughly developed in terms of the following features: ● Modeling constructs for developing schemas for database applications. ● Constraints facilities for expressing certain types of relationships and constraints on the data as determined by application semantics. ● Operations and language facilities to manipulate the database. Out of these three models, the ER model has been primarily employed in CASE tools that are used for database and software design, whereas the other two models have been used as the basis for commercial DBMSs. This unit discusses the emerging class of commercial DBMSs that are called object-relational or enhanced relational systems, and some of the conceptual foundations for these
264 systems. These systems are often called object-relational DBMSs (ORDBMSs). They have emerged as a way of enhancing the capabilities of relational DBMSs (RDBMSs) with some of the features that appeared in object-DBMSs (ODBMSs).
With the arrival of the third generation of database management systems, namely Object-Relational Database Management Systems (ORDBMSs) and Object- Oriented Database Management Systems (OODBMSs), the two disciplines have been combined to allow the concurrent modeling of both data and the processes acting upon the data.
The main forces behind the development of extended ORDBMSs stem from the inability of the legacy DBMSs and the basic relational data model as well as the earlier RDBMSs to meet the challenges of new applications. These are primarily in areas that involve a variety of types of data, for example, text in computer-aided desktop publishing; images in satellite imaging or weather forecasting; complex non-conventional data in engineering designs, in the biological genome information, and in architectural drawings; time series data in history of stock market transactions or sales histories; and spatial and geographic data in maps, air/ water pollution data, and traffic data. Hence, there is a clear need to design databases that can develop, manipulate, and maintain the complex objects arising from such applications. 13.1 Unit Objectives After completing this unit, the reader will be able to: ● Illustrate the basic concept of Relational Database. ● Learn about the Relational database designing. ● Discuss the integrity constraints in RDBMS. ● Gain knowledge about the standards of OODBMS.
265 13.2 Basic Concept of

Relational Databases The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier

83% **MATCHING BLOCK 183/202** W

data models such as the network model or the hierarchical model. The

relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a "flat" file of records.
When a relation is thought of as a table of values, each row in the table represents a collection of related data values.
In the relational model,
each row in the table represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help
in interpreting
the meaning of

86% **MATCHING BLOCK 194/202** SA DBMS_AIML_FINAL.pdf (D111167082)

the values in each row. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is called a domain.

An RDBMS requires only that the database be perceived by the user as tables. Note, however, that this perception applies only to the logical structure of the database: that is, the external and conceptual levels of the ANSI-SPARC architecture. It does not apply to the physical structure of the database, which can be implemented using a variety of storage structures.
In the relational model, relations are used to hold information about the objects to be represented in the database. A relation is represented as a two- dimensional table in which the rows of the table correspond to individual records and the table columns correspond to attributes. Attributes can appear in any order and the relation will still be the same relation, and therefore will convey the same meaning.
Domains are an extremely powerful feature of the relational model. Every attribute in a relation is defined on a domain. Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain.
The
266 domain concept is important, because it allows the user to define in a central place the meaning and source of values that attributes can hold. As a result, more
information is available to the system when it undertakes the execution of a relational operation, and operations that are semantically incorrect can be avoided.
A relation has the following properties: ● The relation has a name that is distinct from all other relation names in the relational schema. ● Each cell of the
relation contains exactly one atomic (single) value. ● Each attribute has a distinct name. ●
The
values of an attribute are all from the same domain. ● Each tuple is distinct; there are no duplicate tuples; ●
The order of attributes has no significance; ● The order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples.) 13.3
Relational Database

99% **MATCHING BLOCK 184/202** W

Design The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data. The needs of the users play a central role in the design process. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broad set of issues. These additional aspects of the expected use of the database influence a variety of design choices at the physical, logical, and view levels.
Phases

of Database Designing

97% **MATCHING BLOCK 185/202** W

A high-level data model serves the database designer by providing a conceptual 267 framework in which to specify, in a systematic fashion, the data requirements of the database users, and a database structure that fulfills these requirements. 1. The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

While there are techniques for diagrammatically representing user requirements, in this unit we restrict ourselves to textual descriptions

94% **MATCHING BLOCK 186/202** W

of user requirements. 2. Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The

entity-relationship model,
which we have studied in previous units,
is typically used to represent the conceptual design. Stated in terms of the entity-relationship model, the conceptual schema specifies the entities that are represented in the database, the attributes of the entities, the relationships among the entities, and constraints on the entities and relationships.

97% **MATCHING BLOCK 187/202** W

Typically, the conceptual- design phase results in the creation of an entity-relationship diagram that provides a graphic representation of the schema. 3. A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements. 4. The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. ● In the logical-design phase, the designer maps the high-level conceptual 268 schema onto the implementation data model of the database system that will be used. The implementation data model is typically the relational data model, and this step typically consists of mapping the conceptual schema defined using the entity-relationship model into a relation schema. ● Finally, the designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified.

The physical schema of a database can be changed relatively easily after an application has been built. However, changes to the logical schema are usually harder to carry out, since they may affect a number of queries and updates scattered across application code. It is therefore important to carry out the database design phase with care, before building the rest of the database application. Relational Database Design (RDD) Relational databases differ from other databases in their approach to organizing data and performing transactions. In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions. Relational database design satisfies the ACID (atomicity, consistency, integrity and durability) properties required from a database design. Relational database design mandates the use of a database server in applications for dealing with data management problems. The four stages of an RDD are as follows: ● Relations and attributes: The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities. ● Primary keys: The attribute or set of attributes that help in uniquely identifying a record is
identified and assigned as
the primary key ● Relationships: The relationships between the various tables are established
269 with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are: o
One
to one o One to many

o
Many to many An entity-relationship diagram can be used to depict the entities, their attributes and the relationship between the entities in a diagrammatic way. • Normalization: This is the process of optimizing the database structure. Normalization simplifies the database design to avoid redundancy and confusion.
The different

55% **MATCHING BLOCK 188/202** W

normal forms are as follows: o First normal form o Second normal form o Third normal form o Boyce-Codd normal form o Fifth normal form

By applying a set of rules, a table is normalized into the above normal forms in a linearly progressive fashion. The efficiency of the design gets better with each higher degree of

100% **MATCHING BLOCK 189/202** W

normalization. When we define an E-R diagram carefully, identifying all entities correctly, the relation schemas generated from the E-R diagram should not need much further normalization. However, there can be functional dependencies between attributes of an entity.

100% **MATCHING BLOCK 190/202** W

Most examples of such dependencies arise out of poor E-R diagram design.

100% **MATCHING BLOCK 191/202** W

Similarly, a relationship set involving more than two entity sets may result in a schema that may not be in a desirable normal form. Since most relationship sets are binary, such cases are relatively rare.

Functional

100% **MATCHING BLOCK 192/202** W

dependencies can help us detect poor E-R design. If the generated 270 relation schemas are not in desired normal form, the problem can be fixed in the E-R diagram. That is, normalization can be done formally as part of data modeling. Alternatively, normalization can be left to the designer's intuition during E-R modeling, and can be done formally on the relation schemas generated from the E-R model.

100% **MATCHING BLOCK 193/202** W

If a multivalued dependency holds and is not implied by the corresponding functional dependency, it usually arises from one of the following sources: • A many-to-many relationship set. • A multivalued attribute of an entity set. For a many-to-many relationship set each related entity set has its own schema and there is an additional schema for the relationship set. For a multivalued attribute, a separate schema is created consisting of that attribute and the primary key of the entity set (as in the case of the phone number attribute of the entity set instructor). The universal-relation approach to relational database design starts with an assumption that there is one single relation schema containing all attributes of interest. This single schema defines how users and applications interact with the database. 13.4

Integrity Constraints We have already
discussed the structural part of the relational data model. A data model has two other parts: a manipulative part, defining the types of operation that are allowed on the data, and a set of integrity constraints, which ensure that the data is accurate.
Because

every attribute has an associated domain, there are constraints (called domain constraints) that form restrictions on the set of values allowed for the attributes of relations. In addition, there are two important integrity rules, which are constraints or restrictions that apply to all instances of the database. The two principal rules for the relational model are known as entity integrity
271 and referential integrity. Other types of integrity constraint are
multiplicity, and general constraints. 1.
Entity
Integrity The first integrity rule applies to the primary keys of base relations.
In a base relation, no attribute of a primary key can be null. By definition, a primary key is a minimal identifier that is used to identify tuples uniquely. This means that no subset of the primary key is sufficient to provide unique identification of tuples. If we allow a null for any part of a primary key, we are implying that not all the attributes are needed to distinguish between tuples, which contradicts the definition of the primary key.
By definition, this attribute must be a primary key, but it contains nulls.
Because
this relation is not a base relation, the model allows the primary key to be null. 2.
Referential Integrity The second integrity rule applies to foreign keys. If a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null. 3.
General Constraints Additional rules specified by the users or database administrators of a database that define or constrain some aspect of the enterprise. It is also possible for users to specify additional constraints that the data must satisfy.
Unfortunately, the level of support for general constraints varies from system to system. 13.5
Introduction to ORDBMS We are already aware that
the relational model has a strong theoretical foundation, based on first-order predicate logic. This theory supported the development of SQL, a declarative language that has now become the standard
272 language for defining and manipulating relational databases. Other strengths of the relational model are its simplicity, its suitability for Online Transaction Processing (OLTP), and its support for data independence. However, the relational data model, and relational DBMSs in particular, are not without their disadvantages.
Here are some weaknesses of the RDBMSs: •
Poor representation of “real-world” entities The process of normalization generally leads to the creation of relations that do not correspond to entities in the “real-world.” The fragmentation of a “real-world” entity into many relations, with a physical representation that reflects this structure, is inefficient leading to many joins during query processing. •
Semantic overloading The relational model has only one construct for representing data and relationships between data: the relation. For example, to represent a many- to-many (*:*) relationship between two entities A and B, we create three relations, one to represent each of the entities A and B and one to represent the relationship. There is no mechanism to distinguish between entities and relationships, or to distinguish between different kinds of relationship that exist between entities. For example, a 1:* relationship might be Has , Owns, Manages, and so on. If such distinctions could be made, then it might be possible to
build the semantics into the operations. It is said that the relational model is semantically overloaded. •
Poor support for integrity and general constraints Integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Unfortunately, many commercial systems do not fully support these constraints and it is necessary to build them into the applications. This, of course, is dangerous and can lead to duplication of effort and, worse still, inconsistencies.
Furthermore, there is
273 no support for general constraints in the relational model, which again means
that
they have to be built into the DBMS or the application. •
Homogeneous data structure The relational model assumes both horizontal and vertical homogeneity. Horizontal homogeneity means that each tuple of a relation must be composed of the same attributes. Vertical homogeneity means that the values in a particular column of a relation must all come from the same domain. Additionally, the intersection of a row and column must be an atomic value. This fixed structure is too restrictive for many “real-world” objects that have a complex structure, and it leads to unnatural joins, which are inefficient as mentioned previously. In defense of the relational data model, it could equally be argued that its symmetric structure is one of the model’s strengths. •

Limited operations The relational model has only a fixed set of operations, such as set and tuple-oriented operations, operations that are provided in the SQL specification. However, SQL does not allow new operations to be specified. Again, this is too restrictive to model the behavior of many real-world objects. For example, a GIS application typically uses points, lines, line groups, and polygons, and needs operations for distance, intersection, and containment. • Difficulty handling recursive queries Atomicity of data means that repeating groups are not allowed in the relational model. As a result, it is extremely difficult to handle recursive queries, that is,
queries about relationships that a relation has with itself (directly or indirectly).
Relational DBMSs are currently the dominant
database technology.
Until recently, the choice of DBMS seemed to be between the relational DBMS
274 and the object-oriented DBMS. However, many vendors of RDBMS products are conscious of the threat and promise of the OODBMS. They agree that traditional relational DBMSs are not suited to the advanced applications, and that added functionality is required. However, they reject the claim that extended RDBMSs will not provide sufficient functionality or will be too slow to cope adequately with the new complexity. If we examine the advanced database applications that are emerging, we find they make extensive use of many object-oriented features such as a user-extensible type system, encapsulation, inheritance, polymorphism, dynamic binding of methods, complex objects including non-first normal form objects, and object identity. The most obvious way to remedy the shortcomings of the relational model is to extend the model with these types of features. This is the approach that has been taken by many extended relational DBMSs, although each has implemented different combinations of features. Thus, there is no single extended relational model; rather, there are a variety of these models, whose characteristics depend upon the way and the degree to which extensions were made. However, all the models do share the same basic relational tables and query language, all incorporate some concept of "object," and some have the ability to store methods (or procedures or triggers) as well as data in the database. Various terms have been used for systems that have extended the relational data model. The original term that was used to describe such systems was the Extended Relational DBMS (ERDBMS)
and the term Universal Server or Universal DBMS (UDBMS) has also been used.
However, in recent years the more descriptive term Object-Relational DBMS has been used to indicate that the system incorporates some notion of "object."
Three of the leading RDBMS vendors—Oracle, Microsoft, and IBM—have all extended their systems into ORDBMSs, although the functionality provided by each is slightly different. The concept of the ORDBMS, as a hybrid of the RDBMS and the OODBMS, is very appealing, preserving the wealth of knowledge and experience that has been acquired with the RDBMS—so much so that some
275 analysts predict the ORDBMS will have a 50% larger share of the market than the RDBMS. As might be expected, the standards activity in this area is based on extensions to the SQL standard. The national standards bodies have been working on object extensions to SQL since 1991. These extensions have become part of the SQL standard, with releases in 1999, referred to as SQL:1999, 2003, (
SQL:2003), 2006 with extensions for XML (SQL:2006), 2008 (
SQL:2008) and 2011 (SQL:2011). These releases of the SQL standard are an ongoing attempt to standardize extensions to the relational model and query language.
Advantages of ORDBMSs •
The main advantages of extending the relational data model come from reuse and sharing. Reuse comes from the ability to extend the DBMS server to perform standard functionality centrally, rather than have it coded in each application. These advantages also give rise to increased productivity, both for the developer and for the end-user. • Another obvious advantage is that the extended relational approach preserves the significant body of knowledge and experience that has gone into developing relational applications. This is a significant advantage, as many organizations would find it prohibitively expensive to change. If the new functionality is designed appropriately, this approach should allow organizations to take advantage of the new extensions in an evolutionary way without losing the benefits of current database features and functions. Thus, an ORDBMS could be introduced in an integrative fashion, as proof-of- concept projects. The SQL:2011 standard is designed to be compatible with the SQL2 standard, and so any ORDBMS that complies with SQL:2011 should provide this capability. Disadvantages of ORDBMSs
The ORDBMS approach has the obvious disadvantages of complexity and associated increased costs.
In addition, there
are the
proponents of the relational
276 approach who believe the essential simplicity and purity of the relational model are lost with these types of extension. There are also those who believe that the RDBMS is being extended for what will be a minority of applications that do not achieve optimal performance with current relational technology.

Object-oriented
models and programs deeply combine relationships and encapsulated objects to more closely mirror the real world. This defines broader sets of relationships than those expressed in SQL, and involves functional programs interspersed in the object definitions. In fact, objects are fundamentally not extensions of data, but a completely different concept with far greater power to express real-world relationships and behaviors. 13.6
Summary • Database design mainly involves the design of the database schema. The entity-relationship (E-R) data model is a widely used data model for database design. It provides a convenient graphical representation to view data, relationships, and constraints. •

95%	MATCHING BLOCK 195/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is called a domain. •

Relational databases differ from other databases in their approach to organizing data and performing transactions. In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions. •
The main advantages of extending the relational data model come from reuse and sharing. •
The ORDBMS approach has the obvious disadvantages of complexity and associated increased costs.
In addition,
there are the
proponents of the
277 relational approach who believe the essential simplicity and purity of the relational model are lost with these types of extension. 13.7
Key Terms •
Primary keys:
The attribute or set of attributes that help in uniquely identifying a record is
identified and assigned
as the primary key. • Relationships: The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. •
Domain Constraints: The
constraints that form restrictions on the set of values allowed for the attributes of relations. 13.8
Check Your Progress Short- Answer Type Q1)

95%	MATCHING BLOCK 202/202	SA	DBMS_AIML_FINAL.pdf (D111167082)

A row is called a _____ , a column header is called an _____.

Q2)
The relation has a name that is distinct from all other relation names in the
relational schema. (
True/ False?) Q3) _____ simplifies the database design to avoid redundancy and confusion. Q4)
The attribute or set of attributes that help in uniquely identifying a record is
identified and assigned as the _____ key.
Q5)
Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain. (
True/ False?) Long- Answer Type Q1) What are integrity constraints in Relational DBMS? Explain its types. Q2) Explain the phases of designing in Relational database design.
278 Q3) Give the advantages and disadvantages of ORDBMS. Q4) State the properties of a relation in RDBMS. Q5) Discuss the weaknesses of RDBMS that lead to development of ORDBMS. References •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition. •
Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.

279 Unit: 14 Products and Applications Structure 14.0 Introduction 14.1 Unit Objectives 14.2 Overview of object model of ODMG 14.3 Object Definition & Query Language 14.4 An overview of SQL3 14.5 Nested relations and collections 14.6 Implementation issues for extended type 14.7 Comparing OODBMS & ORDBMS 14.8 Summary 14.9 Key Terms 14.10 Check Your Progress 14.0 Introduction Having a standard for a particular type of database system is very important, because it provides support for portability of database applications. Portability is generally defined as the capability to execute a particular application program on different systems with minimal modifications to the program itself. In the object database field, portability would allow a program written to access one Object- Database Management System (ODBMS) package. This is important to database users because they are generally wary of investing in a new technology if the different vendors do not adhere to a standard. A second potential advantage of having and adhering to standards is that it helps in achieving interoperability, which generally
refers to the ability of an application to access multiple distinct systems.
In database terms, this means
280 that
the same application program may access some data stored under one ODBMS package, and other data stored under another
package. A third advantage of standards is that it
allows customers to compare commercial products more easily by determining which parts of the
standard are supported by each product. The lack of a standard for ODBMSs until recently may have caused some potential users to shy away from converting to this new technology. A consortium of ODBMS vendors, called ODMG (Object Data Management Group), proposed a standard that is known as the ODMG-93 or ODMG 1.0 standard. This was revised into ODMG 2.0 later on. The standard is made up of several parts:
the object model, the object definition language (ODL), the object query language (
OQL), and the bindings to object- oriented programming languages. Language bindings have been specified for three object-oriented programming languages—namely, C++, SMALLTALK, and JAVA. We have already
examined some of the basic concepts of Object- Oriented Database Management Systems (OODBMSs). Here, we will continue our study of these systems and examine the object model and specification languages proposed by the Object Data Management Group (ODMG). The ODMG object model is important because it specifies a standard model for the semantics of database objects and supports interoperability between compliant systems. It became the de facto standard for OODBMSs. To put the discussion of OODBMSs into a commercial context, we also examine the architecture and functionality of ObjectStore, a commercial OODBMS.
This unit also describes the Object Definition Language (ODL), and Object Query Language (OQL). We will also discuss an overview of SQL3 and the Nested Relational systems. 14.1 Unit Objectives After completing this unit, the reader will be able to:
281 • Illustrate the basic concept of Object Data Management Group (ODMG). • Learn about the Object definition language (ODL) and Object Query Language (OQL). • Gain knowledge about SQL3. • Discuss the concept of Nested relational systems and Implementation issues. • Differentiate between OODBMS & ORDBMS. 14.2 Overview of object model of ODMG The ODMG object model is the data model upon which the object definition language (ODL) and object query language (OQL) are based. In fact, this object model provides the data types, type constructors, and other concepts that can be utilized in the ODL to specify object database schemas. Hence, it is meant to provide a standard data model for object-oriented databases, just as the SQL report describes a standard data model for relational databases. It also provides a standard terminology in a field where the same terms were sometimes used to describe different concepts. Objects and literals are the basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has only a value but no object identifier. In either case, the value can have a complex structure. The object state can change over time by modifying the object value.
A literal is basically a constant value, possibly having a complex structure,
that does not change. An object is described by four characteristics: (1) identifier, (2) name, (3) lifetime, and (4) structure. • The object identifier is a unique system-wide identifier (or Object_Id). Every object must have an object identifier.

282 • In addition to the Object_Id, some objects may optionally be given a unique name within a particular database. This name can be used to refer to the object in a program, and the system should be able to locate the object given that name. Obviously, not all individual objects will have unique names. Typically, a few objects, mainly those that hold collections of objects of a particular object type—such as extents—will have a name. These names are used as entry points to the database; that is, by locating these objects by their unique name, the user can then locate other objects that are referenced from these objects. Other important objects in the application may also have unique names. All such names within a particular database must be unique. • The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates). • Finally, the structure of an object specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or a collection object. In the object model, a literal is a value that does not have an object identifier. However, the value may have a simple or complex structure. There are three types of literals: (1)
atomic, (2) collection, and (3) structured. • Atomic literals correspond to the values of basic data types
and are predefined. The basic data types of the object model include long, short, and unsigned integer numbers (these are specified by the keywords Long, Short, Unsigned Long, Unsigned Short in ODL), regular and double precision floating point numbers (Float, Double), boolean values (Boolean), single characters (Char), character strings (String), and enumeration types (Enum), among others. • Structured literals correspond roughly to values that are constructed using the
tuple constructor. They include Date, Interval, Time, and Timestamp as built-in structures,
as well as any additional user-defined type structures as
283 needed by each application. User-defined structures are created using the Struct keyword in ODL, as in the C and C++ programming languages. • Collection literals specify a value that is a collection of objects or values but the collection itself does not have an Object_Id. The collections in the object model are Set>t<, Bag>t<, List>t<, and Array>t<, where t is the type of objects or values in the collection. • The notation of ODMG uses three concepts: interface, literal, and class. Following the ODMG terminology, we use the word behavior to refer to operations and state to refer to properties (attributes and relationships). • An interface specifies only behavior of an object type and is typically non- instantiable (that is, no objects are created corresponding to an interface). Although an interface may have state properties (attributes and relationships) as part of its specifications, these cannot be inherited from the interface. Hence, an interface serves to define operations that can be inherited by other interfaces, as well as by classes that define the user- defined objects for a particular application. • A class specifies both state (attributes) and behavior (operations) of an object type and is instantiable. Hence, database and application objects are typically created based on the user-specified class declarations that form a database schema. • Finally, a literal declaration specifies state but no behavior. Thus, a literal instance holds a simple or complex structured value but has neither an object identifier nor encapsulated operations. • In general, operations are applied to objects using the dot notation. For example, given an object O, to compare it with another object P, we write O.same_as (P) The result returned by this operation is Boolean and would be true if the identity of P is the same as that of O, and false otherwise. Similarly, to create a copy P of
284 object O, we write P = O.copy () An alternative to the dot notation is the arrow notation: O → same_as (P) or O → copy (). In the ODMG object model, two types of inheritance relationships exist: behavior only inheritance and state plus behavior inheritance. Behavior inheritance is also known as ISA or interface inheritance and is specified by the colon (:) notation. Hence, in the ODMG object model, behavior inheritance requires the supertype to be an interface, whereas the subtype could be either a class or another interface. Inheritance in the Object Model of ODMG The other inheritance relationship, called EXTENDS inheritance, is specified by the keyword extends. It is used to inherit both state and behavior strictly among classes, so both the supertype and the subtype must be classes. Multiple inheritance via extends is not permitted. However, multiple inheritance is allowed for behavior inheritance via the colon (:) notation. Hence, an interface may inherit behavior from several other interfaces. A class may also inherit behavior from several interfaces via colon (:) notation, in addition to inheriting behavior and state from at most one other class via extends. 14.3 Object Definition & Query Language After our overview of the ODMG 2.0 object model in the previous section, we now show how these concepts can be utilized to create an object database schema using the object definition language ODL. The ODL is designed to support the semantic constructs of the ODMG 2.0 object model and is independent of any particular programming language. Its main use is to create object specifications— that is, classes and interfaces. Hence, ODL is not a full programming language. A user can specify a database schema in ODL independently of any programming

285 language, then use the specific language bindings to specify how ODL constructs can be mapped to constructs in specific programming languages, such as C++, SMALLTALK, and JAVA. We will describe the concepts of ODL using an example. Figure 14.1 (b) shows a possible object schema for part of the UNIVERSITY database. The graphical notation for Figure 14.1(b) is shown in Figure 14.1 (a) and can be considered as a variation of EER diagrams with the added concept of interface inheritance but without several EER concepts, such as categories (union types) and attributes of relationships. Program 14.1 shows one possible set of ODL class definitions for the UNIVERSITY database. In general, there may be several possible mappings from an object schema diagram (or EER schema diagram) into ODL classes. Program 14.1 shows the straightforward way of mapping part of the UNIVERSITY database. Entity types are mapped into ODL classes, and inheritance is done using extends. However, there is no direct way to map categories (union types) or to do multiple inheritance. In Program 14.1 the classes PERSON, FACULTY, STUDENT, and GRAD_STUDENT have the extents PERSONS, FACULTY, STUDENTS, and GRAD_STUDENTS, respectively. Both FACULTY and STUDENT extends PERSON and GRAD_STUDENT extends STUDENT. Hence, the collection of STUDENTS (and the collection of FACULTY) will be constrained to be a subset of the collection of PERSONs at any time. Similarly, the collection of GRAD_STUDENTs will be a subset of STUDENTs. At the same time, individual STUDENT and FACULTY objects will inherit the properties (attributes and relationships) and operations of PERSON, and individual GRAD_STUDENT objects will inherit those of STUDENT.
286 (a) (b) Figure 14.1 An example of a database schema. (a) Graphical notation for representing ODL schemas. (b) A graphical object database schema for part of the UNIVERSITY database
287 The classes DEPARTMENT, COURSE, SECTION, and CURR_SECTION in Program 14.1 are straightforward mappings of the corresponding entity types in Figure 14.1 (b). Program 14.1: Possible ODL schema for the UNIVERSITY database class PERSON (extent PERSONS key Ssn) { attribute struct Pname { string Fname, string Mname, string Lname } Name; attribute string Ssn; attribute date Birth_date; attribute enum Gender{M, F} Sex; attribute struct Address { short No, string Street, short Apt_no, string City, string State, short Zip } Address; short Age(); }; class FACULTY extends PERSON (extent FACULTY)
288 { attribute string Rank; attribute float Salary; attribute string Office; attribute string Phone; relationship DEPARTMENT Works_in inverse DEPARTMENT::Has faculty; relationship set>GRAD_STUDENT< Advises inverse GRAD_STUDENT::Advisor; relationship set>GRAD_STUDENT<
On_committee_ofinverseGRAD_STUDENT::Committee; void give_raise(in float raise); void promote(in string new rank); }; class GRADE (extent GRADES) { attribute enum GradeValues{A,B,C,D,F,I, P} Grade; relationship SECTION Section inverse SECTION::Students; relationship STUDENT Student inverse STUDENT::Completed_sections; }; class STUDENT extends PERSON (extent STUDENTS) { attribute string Class; attribute Department Minors_in; relationship Department Majors_in inverse DEPARTMENT::Has_majors; relationship set>GRADE< Completed_sections inverse GRADE::Student; relationship set>CURR_SECTION<
289 Registered_in INVERSE CURR_SECTION::Registered_students; void change_major(in string dname) raises(dname_not_valid); float gpa(); void register(in short secno) raises(section_not_valid); void assign_grade(in short secno; IN GradeValue grade) raises(section_not_valid,grade_not_valid); }; class DEGREE { attribute string College; attribute string Degree; attribute string Year; }; class GRAD_STUDENT extends STUDENT (extent GRAD_STUDENTS) { attribute set>Degree< Degrees; relationship Faculty advisor inverse FACULTY::Advises; relationship set>FACULTY< Committee inverse FACULTY::On_committee_of; void assign_advisor(in string Lname; in string Fname) raises(faculty_not_valid); void assign_committee_member(in string Lname; in string Fname) raises(faculty_not_valid); }; class DEPARTMENT (extent DEPARTMENTS key Dname) { attribute string Dname; attribute string Dphone; attribute string Doffice;
290 attribute string College; attribute FACULTY Chair; relationship set>FACULTY< Has_faculty inverse FACULTY::Works_in; relationship set>STUDENT< Has_majors inverse STUDENT::Majors_in; relationship set>COURSE< Offers inverse COURSE::Offered_by; }; class COURSE (extent COURSES key Cno) { attribute string Cname; attribute string Cno; attribute string Description; relationship set>SECTION< Has_sections inverse SECTION::Of_course; relationship >DEPARTMENT< Offered_by inverse DEPARTMENT::Offers; }; class SECTION (extent SECTIONS) { attribute short Sec_no; attribute string Year; attribute enum Quarter{Fall, Winter, Spring, Summer} Qtr; relationship set>Grade< Students inverse Grade::Section; relationship COURSE Of_course inverse COURSE::Has_sections; }; class CURR_SECTION extends SECTION (extent CURRENT_SECTIONS) { relationship set>STUDENT< Registered_students

291 inverse STUDENT::Registered_in void register_student(in string Ssn) raises(student_not_valid, section_full); }; Thus, Multiple inheritance of interfaces by a class is allowed, as is multiple inheritance of interfaces by another interface. However, with extends (class) inheritance, multiple inheritance is not permitted. Hence, a class can inherit from at most one class (in addition to inheriting from zero or more interfaces). The Object Query Language (OQL) The object query language OQL is the query language proposed for the ODMG object model. It is designed to work closely with the programming languages for which an ODMG binding is defined, such as C++, Smalltalk, and Java. Hence, an OQL query embedded into one of these programming languages can return objects that match the type system of that language. Additionally, the implementations of class operations in an ODMG schema can have their code written in these programming languages. The OQL syntax for queries is similar to the syntax of the relational standard query language SQL, with additional features for ODMG concepts, such as object identity, complex objects, operations, inheritance, polymorphism, and relationships. The basic OQL syntax is a select ... from ... where ... structure, as it is for SQL. For example, the query to retrieve the names of all departments in the college of 'Engineering' can be written as follows: Q0: select D.Dname from D in DEPARTMENTS where D.College = 'Engineering';
In general, an entry point to the database is needed for each query, which can be any named persistent object. For many queries, the entry point is the name of the
292 extent of a class. Recall that the extent name is considered to be the name of a persistent object whose type is a collection (in most cases, a set) of objects from the class. Looking at the extent names in Program 14.1, the named object DEPARTMENTS is of type set>DEPARTMENT<; PERSONS is of type set>PERSON<; FACULTY is of type set>FACULTY<; and so on. The use of an extent name—DEPARTMENTS in Q0—as an entry point refers to a persistent collection of objects. Whenever a collection is referenced in an OQL query, we should define an iterator variable—D in Q0—that ranges over each object in the collection. In many cases, as in Q0, the query will select certain objects from the collection, based on the conditions specified in the where clause. In Q0, only persistent objects D in the collection of DEPARTMENTS that satisfy the condition D.College = 'Engineering' are selected for the query result. For each selected object D, the value of D.Dname is retrieved in the query result. Hence, the type of the result for Q0 is bag>string< because the type of each Dname value is string (even though the actual result is a set because Dname is a key attribute). In general, the result of a query would be of type bag for select ... from ... and of type set for select distinct ... from ... , as in SQL (adding the keyword distinct eliminates duplicates). Using the example in Q0, there are three syntactic options for specifying iterator variables: D in DEPARTMENTS DEPARTMENTS D DEPARTMENTS AS D The named objects used as database entry points for OQL queries are not limited to the names of extents. Any named persistent object, whether it refers to an atomic (single) object or to a collection object, can be used as a database entry point. In general, the result of a query can be of any type that can be expressed in the ODMG object model. A query does not have to follow the select ... from ... where
293 ... structure; in the simplest case, any persistent name on its own is a query, whose result is a reference to that persistent object. For example, the query Q1: DEPARTMENTS; returns a reference to the collection of all persistent DEPARTMENT objects, whose type is set>DEPARTMENT<. Similarly, suppose we had given a persistent name CS_DEPARTMENT to a single DEPARTMENT object (the Computer Science department); then, the query Q1A: CS_DEPARTMENT; returns a reference to that individual object of type DEPARTMENT. Once an entry point is specified, the concept of a path expression can be used to specify a path to related attributes and objects. A path expression typically starts at a persistent object name, or at the iterator variable that ranges over individual objects in a collection. This name will be followed by zero or more relationship names or attribute names connected using the dot notation. For example, referring to the UNIVERSITY database in Program 14.1, the following are examples of path expressions, which are also valid queries in OQL: Q2: CS_DEPARTMENT.Chair; Q2A: CS_DEPARTMENT.Chair.Rank; Q2B: CS_DEPARTMENT.Has_faculty; The first expression Q2 returns an object of type FACULTY, because that is the type of the attribute Chair of the DEPARTMENT class. This will be a reference to the FACULTY object that is related to the DEPARTMENT object whose persistent name is CS_DEPARTMENT via the attribute Chair; that is, a reference to the FACULTY object who is chairperson of the Computer Science department. The second expression Q2A is similar, except that it returns the Rank of this FACULTY object (the Computer Science chair) rather than the object reference; hence, the type returned by Q2A is string, which is the data type for the Rank attribute of the FACULTY class.

294 Path expressions Q2 and Q2A return single values, because the attributes Chair (of DEPARTMENT) and Rank (of FACULTY) are both single-valued and they are applied to a single object. The third expression, Q2B, is different; it returns an object of type set>FACULTY< even when applied to a single object, because that is the type of the relationship Has_faculty of the DEPARTMENT class. The collection returned will include a set of references to all FACULTY objects that are related to the DEPARTMENT object whose persistent name is CS_DEPARTMENT via the relationship Has_faculty; that is, a set of references to all FACULTY objects who are working in the Computer Science department. Now, to return the ranks of Computer Science faculty, we cannot write Q3′: CS_DEPARTMENT.Has_faculty.Rank; because it is not clear whether the object returned would be of type set>string< or bag>string< (the latter being more likely, since multiple faculty may share the same rank). Because of this type of ambiguity problem, OQL does not allow expressions such as Q3′. Rather, one must use an iterator variable over any collections, as in Q3A or Q3B below: Q3A: select F.Rank from F in CS_DEPARTMENT.Has_faculty; Q3B: select distinct F.Rank from F in CS_DEPARTMENT.Has_faculty; Here, Q3A returns bag>string< (duplicate rank values appear in the result), whereas Q3B returns set>string< (duplicates are eliminated via the distinct keyword). Both Q3A and Q3B illustrate how an iterator variable can be defined in the from clause to range over a restricted collection specified in the query. The variable F in Q3A and Q3B ranges over the elements of the collection CS_DEPARTMENT.Has_faculty, which is of type set>FACULTY<, and includes only those faculty who are members of the Computer Science department.
295 14.4 An overview of SQL3 SQL (Structured Query Language) was first specified in the 1970s and underwent enhancements in 1989 and 1992. The language is continuing its evolution toward a new standard called SQL3, which adds object-oriented and other features. SQL can be extended to deal simultaneously with tables from the relational model and classes and objects from the object model. The SQL3 standard includes the following parts: • SQL/Framework, SQL/Foundation, SQL/Bindings, SQL/Object. • New parts addressing temporal, transaction aspects of SQL. • SQL/CLI (Call Level Interface). • SQL/PSM (Persistent Stored Modules). SQL/Foundation deals with new data types, new predicates, relational operations, cursors, rules and triggers, user-defined types, transaction capabilities, and stored routines. SQL/CLI (Call Level Interface) provides rules that allow execution of application code without providing source code and avoids the need for preprocessing. It provides a new type of language binding and is analogous to dynamic SQL in SQL-92. Based on Microsoft ODBC (Open Database Connectivity) and SQL Access Group's standard, it contains about 50 routines for tasks such as connection to the SQL server, allocating and deallocating resources, obtaining diagnostic and implementation information, and controlling termination of transactions. SQL/PSM (Persistent Stored Modules) specifies facilities for partitioning an application between a client and a server. The goal is to enhance performance by minimizing network traffic. SQL/Bindings includes Embedded SQL and Direct Invocation as in SQL-92. Embedded SQL has been enhanced to include additional exception declarations. SQL/Temporal deals with historical data, time series data, and other temporal extensions, and it is being proposed by the TSQL2 committee. SQL/Transaction specification formalizes the XA interface for use by SQL implementers.
296 New types of operations have been added to SQL3. These include: • SIMILAR- It allows the use of regular expressions to match character strings. • UNKNOWN- Boolean values have been extended with this operation when a comparison yields neither true nor false because some values may be null. • Linear Recursion- A major new operation is linear recursion for specifying recursive queries. To illustrate this, suppose we have a table called PART_TABLE(Part1, Part2), which contains a tuple >p1, p2< whenever part p1 contains part p2 as a component. A query to produce the bill of materials for some part p1 (that is, all component parts needed to produce p1) is written as a recursive query as follows: WITH RECURSIVE BILL_MATERIAL (Part1, Part2) AS (SELECT Part1, Part2 FROM PART_TABLE WHERE Part1 = 'p1' UNION ALL SELECT PART_TABLE(Part1), PART_TABLE(Part2) FROM BILL_MATERIAL, PART_TABLE WHERE PART_TABLE.Part1 = BILL_MATERIAL(Part2)) SELECT * FROM BILL_MATERIAL ORDER BY Part1, Part2; The final result is contained in BILL_MATERIAL(Part1, Part2). The UNION ALL operation is evaluated by taking a union of all tuples generated by the inner block until no new tuples can be generated. Because SQL2 lacks recursion, it was left to the programmer to accomplish it by appropriate iteration. For security in SQL3, the concept of role is introduced, which is similar to a "job description" and is subject to authorization of privileges. The actual persons (user

297 accounts) that are assigned to a role may change, but the role authorization itself does not have to be changed. SQL3 also includes syntax for the specification and use of triggers as active rules. Triggering events include the INSERT, DELETE, and UPDATE operations on a table. The trigger can be specified to be considered BEFORE or AFTER the triggering event. This feature is present in both of the ORDBMS systems we discussed. The concept of trigger granularity is included in SQL3, which allows the specification of both row-level triggers (the trigger is considered for each affected row) or statement-level trigger (the trigger is considered only once for each triggering event). For distributed (client-server) databases, the concept of a client module is included in SQL3. A client module may contain externally invoked procedures, cursors, and temporary tables, which can be specified using SQL3 syntax. SQL3 also is being extended with programming language facilities. Routines written in computationally complete SQL with full matching of data types and an integrated environment are referred to as SQL routines. To make the language computationally complete, the following programming control structures are included in the SQL3 syntax: CALL/RETURN, BEGIN/END, FOR/END_FOR, IF/THEN/ELSE/END_IF, CASE/END_CASE, LOOP/END_LOOP, WHILE/END_WHILE, REPEAT/UNTIL/END_REPEAT, and LEAVE. Variables are declared using DECLARE, and assignments are specified using SET. External routines refer to programs written in a host language (ADA, C, COBOL, PASCAL, etc.), possibly containing embedded SQL and having possible type mismatches. The advantage of external routines is that there are existing libraries of such routines that are broadly used, which can cut down a lot of implementation effort for applications. On the other hand, SQL routines are more "pure," but they have not been in wide use. SQL routines can be used for server routines (schema-level routines or modules) or as client modules, and they may be procedures or functions that return values. A number of built-in functions enhance the capability of SQL3. They are used to manage handles, which in turn are classified into environment handles that
298 refer to capabilities, connection handles that are connections to servers, and statement handles that manage SQL statements The SQL/Object specification extends SQL-92 to include object-oriented capabilities. New data types include Boolean, character, and binary large objects (LOBs), and large object locators. SQL3 proposes LOB manipulation within the DBMS without having to use external files. Certain operators do not apply to LOB-valued attributes—for example, arithmetic comparisons, group by, and order by. On the other hand, retrieval of partial value, LIKE comparison, concatenation, substring, position, and length are operations that can be applied to LOBs. Under SQL/Foundation and SQL/Object Specification, SQL allows user-defined data types, type constructors, collection types, user-defined functions and procedures, support for large objects, and triggers. Objects in SQL3 are of two types: • Row or tuple types whose instances are tuples in tables. • Abstract Data Types (shortened as ADT or value ADT), which are any general types used as components of tuples. A row type may be defined using the syntax CREATE ROW TYPE row_type_name (>component declarations<); In SQL3 a construct similar to class definition is provided whereby the user can create a named user-defined type with its own behavioral specification and internal structure; it is known as an Abstract Data Type (ADT). The general form of an ADT specification is: CREATE TYPE >type-name< (list of component attributes with individual types declaration of EQUAL and LESS THAN functions declaration of other functions (methods)); An ADT has a number of user-defined functions associated with it. The syntax is
299 FUNCTION >name< (>argument_list<) RETURNS >type<; Two types of functions can be defined: internal SQL3 and external. Internal functions are written in the extended (computationally complete) version of SQL. External functions are written in a host language, with only their signature (interface) appearing in the ADT definition. The form of an external function definition is DECLARE EXTERNAL >function_name< >signature< LANGUAGE >language_name<; Many ORBDMSs have taken the approach of defining a set of ADTs and associated functions for specific application domains, and packaging them together. For example, the Data Blades in Informix Universal Server and the cartridges in Oracle can be considered as such packages or libraries of ADTs for specific application domains. ADTs can be used as the types for attributes in SQL3 and the parameter types in a function or procedure, and as a source type in a distinct type. Type Equivalence is defined in SQL3 at two levels. Two types are name equivalent if and only if they have the same name. Two types are structurally equivalent if and only if they have the same number of components and the components are pairwise type equivalent. Under SQL-92, the definition of UNION-compatibility among two tables is based on the tables being structurally equivalent. Operations on columns, however, are based on name equivalence. Attributes and functions in ADTs are divided into three categories: • PUBLIC (visible at the ADT interface). • PRIVATE (not visible at the ADT interface). • PROTECTED (visible only to subtypes). It is also possible to define virtual attributes as part of ADTs, which are computed and updated using functions. SQL3 has rules for dealing with inheritance (specified via the UNDER keyword), overloading, and resolution of functions. They

300 can be summarized as follows: Inheritance • All attributes are inherited. • The order of supertypes in the UNDER clause determines the inheritance hierarchy. • An instance of a subtype can be used in every context in which a supertype instance is used. Overloading A subtype can redefine any function that is defined in its supertype, with the restriction that the signature be the same. Resolution of Functions • When a function is called, the best match is selected based on the types of all arguments. • For dynamic linking, the runtime types of parameters are considered. • SQL3 supports constructors for collection types, which can be used for creating nested structures for complex objects. List, set, and multiset are supported as built-in type constructors. Arguments for these type constructors can be any other type, including row types, ADTs, and other collection types. Instances of these types can be treated as tables for query purposes. Collections can be unnested by correlating derived tables in SQL3. Another facility in SQL3 is the supertable/subtable facility, which is not equivalent to super and subtypes and no substitutability is assumed. However, a subtable inherits every column from its supertable; every row of a subtable corresponds to one and only one row in the supertable; every row in the supertable corresponds to at most one row in a subtable. INSERT, DELETE, and UPDATE operations are appropriately propagated.
301 14.5 Nested relations and collections The nested relational data model follows the concept of non-normal form relations. No commercial DBMS has chosen to implement this concept in its original form. The nested relational model removes the restriction of the first normal form from the basic relational model, and thus is also known as the Non- 1NF or Non-First Normal Form (NFNF) or NF 2 relational model. In the basic relational model—also called the flat relational model—attributes are required to be single-valued and to have atomic domains. The nested relational model allows composite and multivalued attributes, thus leading to complex tuples with a hierarchical structure. This is useful for representing objects that are naturally hierarchically structured. For example, to define the DEPT schema as a nested structure, we can write the following: DEPT = (DNO, DNAME, MANAGER, EMPLOYEES, PROJECTS, LOCATIONS) EMPLOYEES = (ENAME, DEPENDENTS) PROJECTS = (PNAME, PLOC) LOCATIONS = (DLOC) DEPENDENTS = (DNAME, AGE) First, all attributes of the DEPT relation are defined. Next, any nested attributes of DEPT—namely, EMPLOYEES, PROJECTS, and LOCATIONS—are themselves defined. Next, any second-level nested attributes, such as DEPENDENTS of EMPLOYEES, are defined, and so on. All attribute names must be distinct in the nested relation definition. Notice that a nested attribute is typically a multivalued composite attribute, thus leading to a "nested relation" within each tuple. When a nested relational database schema is defined, it consists of a number of external relation schemas; these define the top level of the individual nested relations. In addition, nested attributes are called internal relation schemas, since they define relational structures that are nested inside another relation. In our example, DEPT is the only external relation. All the others—EMPLOYEES,
302 PROJECTS, LOCATIONS, and DEPENDENTS—are internal relations. Finally, simple attributes appear at the leaf level and are not nested. We can represent each relation schema by means of a tree structure, where the root is an external relation schema, the leaves are simple attributes, and the internal nodes are internal relation schemas. Notice the similarity between this representation and a hierarchical schema. It is important to be aware that the three first-level nested relations in DEPT represent independent information. Hence, EMPLOYEES represents the employees working for the department, PROJECTS represents the projects controlled by the department, and LOCATIONS represents the various department locations. The relationship between EMPLOYEES and PROJECTS is not represented in the schema; this is an M:N relationship, which is difficult to represent in a hierarchical structure. Extensions to the relational algebra and to the relational calculus, as well as to SQL, have been proposed for nested relations. Here, we illustrate two operations, NEST and UNNEST, that can be used to augment standard relational algebra operations for converting between nested and flat relations. Consider the flat EMP_PROJ relation and suppose that we project it over the attributes SSN, PNUMBER, HOURS, ENAME as follows: EMP_PROJ_FLAT ã p SSN, ENAME, PNUMBER, HOURS (EMP_PROJ) To create a nested version of this relation, where one tuple exists for each employee and the (PNUMBER, HOURS) are nested, we use the NEST operation as follows: EMP_PROJ_NESTED ã NEST PROJS = (PNUMBER, HOURS) (EMP_PROJ_FLAT) The effect of this operation is to create an internal nested relation PROJS = (PNUMBER, HOURS) within the external relation EMP_PROJ_NESTED. Hence, NEST groups together the tuples with the same value for the attributes that are not specified in the NEST operation; these are the SSN and ENAME attributes in our example. For each such group, which represents one employee in our

303 example, a single nested tuple is created with an internal nested relation PROJS = (PNUMBER, HOURS). Notice the similarity between nesting and grouping for aggregate functions. In the former, each group of tuples becomes a single nested tuple; in the latter, each group becomes a single summary tuple after an aggregate function is applied to the group. The UNNEST operation is the inverse of NEST. We can reconvert EMP_PROJ_NESTED to EMP_PROJ_FLAT as follows: EMP_PROJ_FLAT ã UNNEST PROJS = (PNUMBER, HOURS) (EMP_PROJ_NESTED) Here, the PROJS nested attribute is flattened into its components PNUMBER, HOURS. 14.6 Implementation Issues for Extended type There are various implementation issues regarding the support of an extended type system with associated functions (operations). • The ORDBMS must dynamically link a user-defined function in its address space only when it is required. As we saw in the case of the two ORDBMSs, numerous functions are required to operate on two-or three-dimensional spatial data, images, text, and so on. With a static linking of all function libraries, the DBMS address space may increase by an order of magnitude. Dynamic linking is available in the two ORDBMSs that we studied. • Client-server issues deal with the placement and activation of functions. If the server needs to perform a function, it is best to do so in the DBMS address space rather than remotely, due to the large amount of overhead. If the function demands computation that is too intensive or if the server is attending to a very large number of clients, the server may ship the function to a separate client machine. For security reasons, it is better to run functions at the client using the user ID of the client. In the future functions are likely to be written in interpreted languages like JAVA.
304 • It should be possible to run queries inside functions. A function must operate the same way whether it is used from an application using the application program interface (API), or whether it is invoked by the DBMS as a part of executing SQL with the function embedded in an SQL statement. Systems should support a nesting of these "callbacks." • Because of the variety in the data types in an ORDBMS and associated operators, efficient storage and access of the data is important. For spatial data or multidimensional data, new storage structures such as R-trees, quad trees, or Grid files may be used. The ORDBMS must allow new types to be defined with new access structures. Dealing with large text strings or binary files also opens up a number of storage and search options. It should be possible to explore such new options by defining new data types within the
ORDBMS. 14.7 Comparing OODBMS & ORDBMS We conclude our treatment of object-relational DBMSs and object-oriented DBMSs with a brief comparison of the two types of system. For the purposes of the comparison, we examine the systems from three perspectives: data modeling (Table 14.1), data access (Table 14.2), and data sharing (Table 14.3). We assume that future ORDBMSs will be compliant with the SQL:2011 standard. Table 14.1 Data modeling comparison of ORDBMS and OODBMS. FEATURE ORDBMS OODBMS Object identity (OID) Supported through REF type Supported Encapsulation Supported through UDTs Supported but broken for
305 queries Inheritance Supported (separate hierarchies for UDTs and tables) Supported Polymorphism Supported (UDF invocation based on the generic function) Supported as in an
object- oriented
programming model language Complex objects Supported through UDTs Supported Relationships Strong support with user- defined referential integrity constraints Supported (for example, using class libraries) Table 14.2 Data access comparison of ORDBMS and OODBMS. FEATURE ORDBMS OODBMS Creating and accessing persistent data Supported but not transparent Supported but degree of transparency differs between products Ad hoc query facility Strong support Supported through ODMG 3.0 Navigation Supported by REF type Strong support Integrity constraints Strong support No support
306 Object server/page server Object server Either Schema evolution Limited support Supported but degree of support differs between products Table 14.3 Data sharing comparison of ORDBMS and OODBMS. FEATURE ORDBMS OODBMS ACID transactions Strong support Supported Recovery Strong support Supported but degree of support differs between products Advanced transaction models No support Supported but degree of support differs between products Security, integrity, and views Strong support Limited support 14.8
Summary • The ODMG object model is the data model upon which the object definition language (ODL) and object query language (OQL) are based. • Objects and literals are the basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has only a value but no object identifier. • An object is described by four characteristics: (1) identifier, (2) name, (3) lifetime, and (4) structure. There are three types of literals: (1) atomic, (2)

307 collection, and (3) structured. • In the ODMG object model, two types of inheritance relationships exist: behavior only inheritance and state plus behavior inheritance. • The ODL is designed to support the semantic constructs of the ODMG 2.0 object model and is independent of any particular programming language. Its main use is to create object specifications—that is, classes and interfaces. • The nested relational model removes the restriction of the first normal form from the basic relational model, and thus is also known as the Non-1NF or Non-First Normal Form (NFNF) or NF 2 relational model. 14.9 Key Terms • Atomic literals: They correspond to the values of basic data types and are predefined. • Structured literals: They correspond roughly to values that are constructed using the tuple constructor. • Collection literals: They specify a value that is a collection of objects or values but the collection itself does not have an Object_Id. • SQL Routines: Routines written in computationally complete SQL with full matching of data types and an integrated environment are referred to as SQL routines. 14.10 Check Your Progress Short- Answer Type Q1) Full form of ODMG- a) Object Data Merging Goal b) Object Data Management Goal
308 c) Object Data Merging Guide d) Object Data Management Group Q2) A major new operation in SQL3 is _____ for specifying recursive queries. Q3) _____ literals correspond to the values of basic data types and are predefined. Q4) The nested relational data model is also known as the Non-1NF data model. (True/ False?) Q5) ORDBMS supports advanced transaction models for data sharing. (True/ False?) Long- Answer Type Q1) Explain what the following terms mean in object-oriented database terminology: method, signature, message, collection, extent. Q2) What are the differences and similarities between objects and literals in the ODMG object model?
Q3) Give a brief overview of SQL3. Q4) Write a short note on the nested relational data model. Q5) Differentiate between OODBMS and ORDBMS. References • Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007. •
Database Systems: A Practical Approach to Design, Implementation, and Management,
Thomas Connolly & Carolyn Begg,
Pearson, 6th Edition.
309
MODULE VIII ADVANCED DATABASES
310 Unit: 15 Introduction
to Advanced Databases Structure 15.0 Introduction 15.1 Unit Objectives 15.2 Active database 15.3 Applications of active database 15.4 Temporal database 15.5 Spatial database 15.6 Summary 15.7 Key Terms 15.8 Check Your Progress 15.0 Introduction As the use of database systems has grown, users have demanded additional functionality from these software packages, with the purpose of making it easier to implement more advanced and complex user applications. Object-oriented databases and object-relational systems do provide features that allow users to extend their systems by specifying additional abstract data types for each application. However, it is quite useful to identify certain common features for some of these advanced applications and to create models that can represent these common features. In addition, specialized storage structures and indexing methods can be implemented to improve the performance of these common features. These features can then be implemented as abstract data type or class libraries and separately purchased with the basic DBMS software package. Users can utilize these features directly if they are suitable for their applications, without having to reinvent, reimplement, and reprogram such common features.
311 This unit introduces database concepts for some of the common features that are needed by advanced applications and that are starting to have widespread use. The features we will cover are active rules that are used in active database applications, temporal concepts that are used in temporal database applications, and spatial database concepts. 15.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the advanced database concepts for active databases. • Illustrate the active rules used in designing. • Discuss the concept of the temporal database and its applications. • Gain knowledge about the spatial databases. 15.2 Active database Rules that specify actions that are automatically triggered by certain events have been considered as important enhancements to a database system for quite some time. In fact, the concept of triggers—a technique for specifying certain types of active rules—has existed in early versions of the SQL specification for relational databases. Commercial relational DBMSs—such as Oracle, DB2, and SYBASE— have had various versions of triggers available. However, much research into what a general model for active databases should look like has been done since the early models of triggers were proposed. The model that has been used for specifying active database rules is referred to as the Event-Condition-Action, or ECA model. A rule in the ECA model has three components: 1. The event (or events) that trigger the rule: These events are usually database update operations that are explicitly applied to the database.

312 However, in the general model, they could also be temporal events or other kinds of external events. 2. The condition that determines whether the rule action should be executed: Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed. 3. The action to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed. Let us consider some examples to illustrate these concepts. The examples are based on a much simplified variation of the COMPANY database application and are shown in Figure 15.1, with each employee having a name (Name), Social Security number (Ssn), salary (Salary), department to which she is currently assigned (Dno, a foreign key to DEPARTMENT), and a direct supervisor (Supervisor_ssn, a (recursive) foreign key to EMPLOYEE). For this example, we assume that NULL is allowed for Dno, indicating that an employee may be temporarily unassigned to any department. Each department has a name (Dname), number (Dno), the total salary of all employees assigned to the department (Total_sal), and a manager (Manager_ssn, which is a foreign key to EMPLOYEE). Figure 15.1 A simplified COMPANY database used for active rule examples.
313 It should be noted that the Total_sal attribute is really a derived attribute whose value should be the sum of the salaries of all employees who are assigned to the particular department. Maintaining the correct value of such a derived attribute can be done via an active rule. First we have to determine the events that may cause a change in the value of Total_sal, which are as follows: 1. Inserting (one or more) new employee tuples 2. Changing the salary of (one or more) existing employees 3. Changing the assignment of existing employees from one department to another 4. Deleting (one or more) employee tuples In the case of event 1, we only need to recompute Total_sal if the new employee is immediately assigned to a department—that is, if the value of the Dno attribute for the new employee tuple is not NULL (assuming NULL is allowed for Dno). Hence, this would be the condition to be checked. A similar condition could be checked for event 2 (and 4) to determine whether the employee whose salary is changed (or who is being deleted) is currently assigned to a department. For event 3, we will always execute an action to maintain the value of Total_sal correctly, so no condition is needed (the action is always executed). The action for events 1, 2, and 4 is to automatically update the value of Total_sal for the employee's department to reflect the newly inserted, updated, or deleted employee's salary. In the case of event 3, a two-fold action is needed: one to update the Total_sal of the employee's old department and the other to update the Total_sal of the employee's new department. The four active rules (or triggers) R1, R2, R3, and R4—corresponding to the above situation—can be specified in the notation of the Oracle DBMS as shown in Figure 15.2(a). Let us consider rule R1 to illustrate the syntax of creating triggers in Oracle. The CREATE TRIGGER statement specifies a trigger (or active rule) name—Total_sal1 for R1. The AFTER clause specifies that the rule will be triggered after the events that trigger the rule occur. The triggering events—an
314 insert of a new employee in this example—are specified following the AFTER keyword. The ON clause specifies the relation on which the rule is specified— EMPLOYEE for R1. The optional keywords FOR EACH ROW specify that the rule will be triggered once for each row that is affected by the triggering event. The optional WHEN clause is used to specify any conditions that need to be checked after the rule is triggered, but before the action is executed. Finally, the action(s) to be taken is (are) specified as a PL/SQL block, which typically contains one or more SQL statements or calls to execute external procedures. The four triggers (active rules) R1, R2, R3, and R4 illustrate a number of features of active rules. First, the basic events that can be specified for triggering the rules are the standard SQL update commands: INSERT, DELETE, and UPDATE. In the case of UPDATE, one may specify the attributes to be updated—for example, by writing UPDATE OF Salary, Dno. Second, the rule designer needs to have a way to refer to the tuples that have been inserted, deleted, or modified by the triggering event. The keywords NEW and OLD are used in Oracle notation; NEW is used to refer to a newly inserted or newly updated tuple, whereas OLD is used to refer to a deleted tuple or to a tuple before it was updated. Thus, rule R1 is triggered after an INSERT operation is applied to the EMPLOYEE relation. In R1, the condition (NEW.Dno IS NOT NULL) is checked, and if it evaluates to true, meaning that the newly inserted employee tuple is related to a department, then the action is executed. The action updates the DEPARTMENT tuple(s) related to the newly inserted employee by adding their salary (NEW.Salary) to the Total_sal attribute of their related department. Rule R2 is similar to R1, but it is triggered by an UPDATE operation that updates the SALARY of an employee rather than by an INSERT. Rule R3 is triggered by an update to the Dno attribute of EMPLOYEE, which signifies changing an employee's assignment from one department to another. There is no condition to check in R3, so the action is executed whenever the triggering event occurs. The action updates both the old department and new department of the reassigned employees by adding their salary to Total_sal of their new department and
315 subtracting their salary from Total_sal of their old department. Note that this should work even if the value of Dno is NULL, because in this case no department will be selected for the rule action. Figure 15.1 Specifying active rules as triggers in Oracle notation. (a) Triggers for automatically maintaining the consistency of Total_sal of DEPARTMENT. (b) Trigger for comparing an employee's salary with that of his or her supervisor.

316 It is important to note the effect of the optional FOR EACH ROW clause, which signifies that the rule is triggered separately for each tuple. This is known as a row-level trigger. If this clause was left out, the trigger would be known as a statement-level trigger and would be triggered once for each triggering statement. To see the difference, consider the following update operation, which gives a 10% raise to all employees assigned to department 5. This operation would be an event that triggers rule R2: UPDATE EMPLOYEE SET Salary = 1.1 * Salary WHERE Dno = 5; There are some additional issues concerning how rules are designed and implemented. The first issue concerns activation, deactivation, and grouping of rules. In addition to creating rules, an active database system should allow users to activate, deactivate, and drop rules by referring to their rule names. A deactivated rule will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed. The activate command will make the rule active again. The drop command deletes the rule from the system. Another option is to group rules into named rule sets, so the whole set of rules can be activated, deactivated, or dropped. It is also useful to have a command that can trigger a rule or rule set via an explicit PROCESS RULES command issued by the user. The second issue concerns whether the triggered action should be executed before, after, instead of, or concurrently with the triggering event. A before trigger executes the trigger before executing the event that caused the trigger. It can be used in applications such as checking for constraint violations. An after trigger executes the trigger after executing the event, and it can be used in applications such as maintaining derived data and monitoring for specific events and conditions. An instead of trigger executes the trigger instead of executing the event, and it can be used in applications such as executing corresponding
317 updates on base relations in response to an event that is an update of a view. Let us assume that the triggering event occurs as part of a transaction execution. We should first consider the various options for how the triggering event is related to the evaluation of the rule's condition. The rule condition evaluation is also known as rule consideration, since the action is to be executed only after considering whether the condition evaluates to true or false. There are three main possibilities for rule consideration: 1. Immediate consideration: The condition is evaluated as part of the same transaction as the triggering event and is evaluated immediately. This case can be further categorized into three options: • Evaluate the condition before executing the triggering event. • Evaluate the condition after executing the triggering event. • Evaluate the condition instead of executing the triggering event. 2. Deferred consideration: The condition is evaluated at the end of the transaction that included the triggering event. In this case, there could be many triggered rules waiting to have their conditions evaluated. 3. Detached consideration: The condition is evaluated as a separate transaction, spawned from the triggering transaction. The next set of options concerns the relationship between evaluating the rule condition and executing the rule action. Here, again, three options are possible: immediate, deferred, or detached execution. Most active systems use the first option. That is, as soon as the condition is evaluated, if it returns true, the action is immediately executed. One of the difficulties that may have limited the widespread use of active rules, in spite of their potential to simplify database and software development, is that there are no easy-to-use techniques for designing, writing, and verifying rules. For example, it is difficult to verify that a set of rules is consistent, meaning that two or more rules in the set do not contradict one another. It is also difficult to
318 guarantee termination of a set of rules under all circumstances. To illustrate the termination problem briefly, consider the rules in Figure 15.2. Here, rule R1 is triggered by an INSERT event on TABLE1 and its action includes an update event on Attribute1 of TABLE2. However, rule R2's triggering event is an UPDATE event on Attribute1 of TABLE2, and its action includes an INSERT event on TABLE1. In this example, it is easy to see that these two rules can trigger one another indefinitely, leading to nontermination. However, if dozens of rules are written, it is very difficult to determine whether termination is guaranteed or not. Figure 15.2 An example to illustrate the termination problem for active rules. If active rules are to reach their potential, it is necessary to develop tools for the design, debugging, and monitoring of active rules that can help users design and debug their rules. 15.3 Applications of Active Database We now briefly discuss some of the potential applications of active rules. Obviously, one important application is to allow notification of certain conditions that occur. For example, an active database may be used to monitor, say, the temperature of an industrial furnace. The application can periodically insert in the database the temperature reading records directly from temperature sensors, and active rules can be written that are triggered whenever a temperature record is inserted, with a condition that checks if the temperature exceeds the danger level and results in the action to raise an alarm.

319 Active rules can also be used to enforce integrity constraints by specifying the types of events that may cause the constraints to be violated and then evaluating appropriate conditions that check whether the constraints are actually violated by the event or not. Hence, complex application constraints, often known as business rules, may be enforced that way. For example, in the UNIVERSITY database application, one rule may monitor the GPA of students whenever a new grade is entered, and it may alert the advisor if the GPA of a student falls below a certain threshold; another rule may check that course prerequisites are satisfied before allowing a student to enroll in a course; and so on. Other applications include the automatic maintenance of derived data, such as the examples of rules R1 through R4 that maintain the derived attribute Total_sal whenever individual employee tuples are changed. A similar application is to use active rules to maintain the consistency of materialized views whenever the base relations are modified. Alternatively, an update operation specified on a view can be a triggering event, which can be converted to updates on the base relations by using an instead of trigger. These applications are also relevant to the new data warehousing technologies. A related application maintains that replicated tables are consistent by specifying rules that modify the replicas whenever the master table is modified. 15.4 Temporal Database Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. Hence, they provide a good example to illustrate the need for developing a set of unifying concepts for application developers to use. Temporal database applications have been developed since the early days of database usage. However, in creating these applications, it is mainly left to the application designers and developers to discover, design, program, and implement the temporal concepts they need. There are many examples of applications where some aspect of time is needed to
320 maintain the information in a database. These include healthcare, where patient histories need to be maintained; insurance, where claims and accident histories are required as well as information about the times when insurance policies are in effect; reservation systems in general (hotel, airline, car rental, train, and so on), where information on the dates and times when reservations are in effect are required; scientific databases, where data collected from experiments includes the time when each data is measured; and so on. In fact, it is realistic to conclude that the majority of database applications have some temporal information. However, users often attempt to simplify or ignore temporal aspects because of the complexity that they add to their applications. For temporal databases, time is considered to be an ordered sequence of points in some granularity that is determined by the application. For example, suppose that some temporal application never requires time units that are less than one second. Then, each time point represents one second using this granularity. In reality, each second is a (short) time duration, not a point, since it may be further divided into milliseconds, microseconds, and so on. The main consequence of choosing a minimum granularity—say, one second—is that events occurring within the same second will be considered to be simultaneous events, even though in reality they may not be. Because there is no known beginning or ending of time, one needs a reference point from which to measure specific time points. Various calendars are used by various cultures (such as Gregorian (Western), Chinese, Islamic, Hindu, Jewish, Coptic, and so on) with different reference points. A calendar organizes time into different time units for convenience. Most calendars group 60 seconds into a minute, 60 minutes into an hour, 24 hours into a day (based on the physical time of earth's rotation around its axis), and 7 days into a week. Further groupings of days into months and months into years either follow solar or lunar natural phenomena and are generally irregular. In the Gregorian calendar, which is used in most Western countries, days are grouped into months that are 28, 29, 30, or 31 days, and 12 months are grouped into a year. Complex formulas are used to
321 map the different time units to one another. Event Information versus Duration (or State) Information A temporal database will store information concerning when certain events occur, or when certain facts are considered to be true. There are several different types of temporal information. Point events or facts are typically associated in the database with a single time point in some granularity. For example, a bank deposit event may be associated with the timestamp when the deposit was made, or the total monthly sales of a product (fact) may be associated with a particular month (say, October 2017). It should be noted that even though such events or facts may have different granularities, each is still associated with a single time value in the database. Duration, events, or facts, on the other hand, are associated with a specific time period in the database. For example, an employee may have worked in a company from September 12, 2011 until July 8, 2013. A time period is represented by its start and end time points [START-TIME, ENDTIME]. For example, the above period is represented as [2011-09-12, 2013- 07-08]. Such a time period is often interpreted to mean the set of all time points from starttime to end-time, inclusive, in the specified granularity. Hence, assuming day granularity, the period [2011-09-12, 2013-07-08] represents the set of all days from September 12, 2011 until July 8, 2013, inclusive. Valid Time and Transaction Time Dimensions Given a particular event or fact that is associated with a particular time point or time period in the database, the association may be interpreted to mean different things. The most natural interpretation is that the associated time is the time that the event occurred, or the period during which the fact was considered to be true in the real world. If this interpretation is used, the associated time is often referred to as the valid time. A temporal database using this interpretation is called a valid time database. However, a different interpretation can be used, where the associated time refers

322 to the time when the information was actually stored in the database; that is, it is the value of the system time clock when the information is valid in the system. In this case, the associated time is called the transaction time. A temporal database using this interpretation is called a transaction time database. Other interpretations can also be intended, but these are considered to be the most common ones, and they are referred to as time dimensions. In some applications, only one of the dimensions is needed and in other cases both time dimensions are required, in which case the temporal database is called a bitemporal database. If other interpretations are intended for time, the user can define the semantics and program the applications appropriately, and this interpretation of time is called a user-defined time. We can incorporate time in relational databases using tuple versioning while in object-oriented databases using attribute versioning. In the tuple versioning approach, whenever one attribute value is changed, a whole new tuple version is created, even though all the other attribute values will be identical to the previous tuple version. An alternative approach can be used in database systems that support complex structured objects, such as object databases or object- relational systems. This approach is called attribute versioning. In attribute versioning, a single complex object is used to store all the temporal changes of the object. Each attribute that changes over time is called a time- varying attribute, and it has its values versioned over time by adding temporal periods to the attribute. The temporal periods may represent valid time, transaction time, or bitemporal, depending on the application requirements. Attributes that do not change over time are called non-time-varying and are not associated with the temporal periods. 15.5 Spatial Database Spatial databases incorporate functionality that provides support for databases that keep track of objects in a multidimensional space. For example, cartographic
323 databases that store maps include two-dimensional spatial descriptions of their objects— from countries and states to rivers, cities, roads, seas, and so on. The systems that manage geographic data and related applications are known as geographic information systems (GISs), and they are used in areas such as environmental applications, transportation systems, emergency response systems, and battle management. Other databases, such as meteorological databases for weather information, are three-dimensional, since temperatures and other meteorological information are related to three-dimensional spatial points. In general, a spatial database stores objects that have spatial characteristics that describe them and that have spatial relationships among them. The spatial relationships among the objects are important, and they are often needed when querying the database. Although a spatial database can in general refer to an n- dimensional space for any n, we will limit our discussion to two dimensions as an illustration. A spatial database is optimized to store and query data related to objects in space, including points, lines and polygons. Satellite images are a prominent example of spatial data. Queries posed on these spatial data, where predicates for selection deal with spatial parameters, are called spatial queries. Table 15.1 shows the common analytical operations involved in processing geographic or spatial data. Measurement operations are used to measure some global properties of single objects (such as the area, the relative size of an object's parts, compactness, or symmetry) and to measure the relative position of different objects in terms of distance and direction. Spatial analysis operations, which often use statistical techniques, are used to uncover spatial relationships within and among mapped data layers. An example would be to create a map—known as a prediction map—that identifies the locations of likely customers for particular products based on the historical sales and demographic information. Flow analysis operations help in determining the shortest path between two points and also the connectivity among nodes or regions in a graph. Location analysis
324 aims to find if the given set of points and lines lie within a given polygon (location). The process involves generating a buffer around existing geographic features and then identifying or selecting features based on whether they fall inside or outside the boundary of the buffer. Digital terrain analysis is used to build three-dimensional models, where the topography of a geographical location can be represented with an x, y, z data model known as Digital Terrain (or Elevation) Model (DTM/DEM). The x and y dimensions of a DTM represent the horizontal plane, and z represents spot heights for the respective x, y coordinates. Such models can be used for analysis of environmental data or during the design of engineering projects that require terrain information. Spatial search allows a user to search for objects within a particular spatial region. There are also topological relationships among spatial objects. These are often used in Boolean predicates to select objects based on their spatial relationships. Table 15.1 Common Types of Analysis for Spatial Data Analysis Type Type of Operations and Measurements Measurements Distance, perimeter, shape, adjacency, and direction Spatial analysis/statistics Pattern, autocorrelation, and indexes of similarity and topology using spatial and nonspatial data Flow analysis Connectivity and shortest path Location analysis Analysis of points and lines within a polygon Terrain analysis Slope/aspect, catchment area, drainage network

325 Spatial Data Types and Models Spatial data comes in three basic forms. These forms have become a de facto standard due to their wide use in commercial systems. • Map data includes various geographic or spatial features of objects in a map, such as an object's shape and the location of the object within the map. The three basic types of features are points, lines, and polygons (or areas). Points are used to represent spatial characteristics of objects whose locations correspond to a single 2-D coordinate (x, y, or longitude/latitude) in the scale of a particular application. Depending on the scale, some examples of point objects could be buildings, cellular towers, or stationary vehicles. Moving vehicles and other moving objects can be represented by a sequence of point locations that change over time. Lines represent objects having length, such as roads or rivers, whose spatial characteristics can be approximated by a sequence of connected lines. Polygons are used to represent spatial characteristics of objects that have a boundary, such as countries, states, lakes, or cities. Notice that some objects, such as buildings or cities, can be represented as either points or polygons, depending on the scale of detail. • Attribute data is the descriptive data that GIS systems associate with map features. For example, suppose that a map contains features that represent counties within a U.S. state (such as Texas or Oregon). Attributes for each county feature (object) could include population, largest city/town, area in square miles, and so on. Other attribute data could be included for other features in the map, such as states, cities, congressional districts, census tracts, and so on. • Image data includes data such as satellite images and aerial photographs, which are typically created by cameras. Objects of interest, such as buildings and roads, can be identified and overlaid on these images. Images can also be attributes of map features. One can add images to other map features so that clicking on the feature would display the image. Aerial and satellite
326 images are typical examples of raster data. Models of spatial information are sometimes grouped into two broad categories: field and object. A spatial application (such as remote sensing or highway traffic control) is modeled using either a field- or an object-based model, depending on the requirements and the traditional choice of model for the application. Field models are often used to model spatial data that is continuous in nature, such as terrain elevation, temperature data, and soil variation characteristics, whereas object models have traditionally been used for applications such as transportation networks, land parcels, buildings, and other objects that possess both spatial and non-spatial attributes. Spatial Operators and Spatial Queries Spatial operators are used to capture all the relevant geometric properties of objects embedded in the physical space and the relations between them, as well as to perform spatial analysis. Operators are classified into three broad categories. • Topological operators. Topological properties are invariant when topological transformations are applied. These properties do not change after transformations like rotation, translation, or scaling. Topological operators are hierarchically structured in several levels, where the base level offers operators the ability to check for detailed topological relations between regions with a broad boundary, and the higher levels offer more abstract operators that allow users to query uncertain spatial data independent of the underlying geometric data model. Examples include open (region), close (region), and inside (point, loop). • Projective operators. Projective operators, such as convex hull, are used to express predicates about the concavity/convexity of objects as well as other spatial relations (for example, being inside the concavity of a given object). • Metric operators. Metric operators provide a more specific description of the object's geometry. They are used to measure some global properties of single
327 objects (such as the area, relative size of an object's parts, compactness, and symmetry), and to measure the relative position of different objects in terms of distance and direction. Examples include length (arc) and distance (point, point). Dynamic Spatial Operators: The operations performed by the operators mentioned above are static, in the sense that the operands are not affected by the application of the operation. For example, calculating the length of the curve has no effect on the curve itself. Dynamic operations alter the objects upon which the operations act. The three fundamental dynamic operations are create, destroy, and update. A representative example of dynamic operations would be updating a spatial object that can be subdivided into translate (shift position), rotate (change orientation), scale up or down, reflect (produce a mirror image), and shear (deform). Spatial Queries: Spatial queries are requests for spatial data that require the use of spatial operations. The following categories illustrate three typical types of spatial queries: 1. Range queries. Find all objects of a particular type that are within a given spatial area; for example, find all hospitals within the Metropolitan Atlanta city area. A variation of this query is to find all objects within a particular distance from a given location; for example, find all ambulances within a five mile radius of an accident location. 2. Nearest neighbor queries. Finds an object of a particular type that is closest to a given location; for example, find the police car that is closest to the location of a crime. This can be generalized to find the k nearest neighbors, such as the 5 closest ambulances to an accident location. 3. Spatial joins or overlays. Typically joins the objects of two types based on some spatial condition, such as the objects intersecting or overlapping spatially or being within a certain distance of one another. For example, find all townships located on a major highway between two cities or find all
328 homes that are within two miles of a lake. The first example spatially joins township objects and highway object, and the second example spatially joins lake objects and home objects. Applications
of Spatial Data Spatial data management is useful in many disciplines, including geography, remote sensing, urban planning, and natural resource management.

Spatial database management is playing an important role in the solution of challenging scientific problems such as global climate change and genomics. Due to the spatial nature of genome data, GIS and spatial database management systems have a large role to play in the area of bioinformatics. Some of the typical applications include pattern recognition (for example, to check if the topology of a particular gene in the genome is found in any other sequence feature map in the database), genome browser development, and visualization maps. Another important application area of spatial data mining is the spatial outlier detection. A spatial outlier is a spatially referenced object whose nonspatial attribute values are significantly different from those of other spatially referenced objects in its spatial neighborhood. Detecting spatial outliers is useful in many applications of geographic information systems and spatial databases. These application domains include transportation, ecology, public safety, public health, climatology, and location-based services. 15.6 Summary • The model that has been used for specifying active database rules is referred to as the Event-Condition-Action, or ECA model. • Time plays an important role in database systems. Databases are models of the real world. Whereas most databases model the state of the real world at a point in time (at the current time), temporal databases model the states of the real world across time.
329 • Facts in temporal relations have associated times when they are valid, which can be represented as a union of intervals. Temporal query languages simplify modeling of time, as well as time-related queries. • Spatial databases are finding increasing use today to store computer-aided design data as well as geographic data. • Spatial databases incorporate functionality that provides support for databases that keep track of objects in a multidimensional space. 15.7 Key Terms • Valid time: The most natural interpretation is that the associated time is the time that the event occurred, or the period during which the fact was considered to be true in the real world. If this interpretation is used, the associated time is often referred to as the valid time. • Transaction time: The associated time refers to the time when the information was actually stored in the database; that is, it is the value of the system time clock when the information is valid in the system. In this case, the associated time is called the transaction time. • Spatial queries: Queries posed on these spatial data, where predicates for selection deal with spatial parameters, are called spatial queries. • Bitemporal database: In some applications, only one of the dimensions is needed and in other cases both time dimensions are required, in which case the temporal database is called a bitemporal database. 15.8 Check Your Progress Short- Answer Type Q1) _____ model has been used for specifying active database rules. Q2) Active databases are based on the concept of-
330 a) Triggers b) Cells c) Pointers d) None of the above Q3) When the rule is triggered separately for each tuple, it is known as a Row- level Trigger. (True/ False?) Q4) Each attribute that changes over time is called a _____ attribute. Q5) The systems that manage geographic data and related applications are known as geographic information systems (GISs). (True/ False?) Long- Answer Type Q1) What are the differences between row-level and statement-level active rules? Q2) Discuss some applications of active databases. Q3) What are the differences among valid time, transaction time, and bitemporal relations? Q4) What are the different types of spatial data? Q5) State the main differences between tuple versioning and attribute versioning? References • Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007. • Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
331 Unit: 16 Multimedia databases Structure 16.0 Introduction 16.1 Unit Objectives 16.2 Concept of multimedia databases 16.3 Automatic Analysis of Images 16.4 Object Recognition in Images 16.5 Semantic Tagging of Images 16.6 Analysis of Audio Data Sources 16.7 Introduction to Mobile Databases 16.8 Summary 16.9 Key Terms 16.10 Check Your Progress 16.0 Introduction Multimedia data is an increasingly popular form of data—are today almost always stored outside the database, in file systems. This kind of storage is not a problem when the number of multimedia objects is relatively small, since features provided by databases are usually not important. However, database features become important when the number of multimedia objects stored is large. Issues such as transactional updates, querying facilities, and indexing then become important. Multimedia objects often have descriptive attributes, such as those indicating when they were created, who created them, and to what category they belong. One approach to building a database for such multimedia objects is to use databases for storing the descriptive attributes and for keeping track of the files in which the multimedia objects are stored.

332 This unit discusses the basic concept of Multimedia databases. We will also discuss automatic analysis of images, object recognition in images, and semantic tagging of images. A brief introduction to mobile databases is also discussed in this unit. 16.1 Unit Objectives After completing this unit, the reader will be able to: • Learn about the concepts of multimedia databases. • Illustrate the various multimedia data formats. • Discuss the analysis of multimedia sources like images. • Gain knowledge about the mobile databases. 16.2 Concept of Multimedia Databases Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images (such as pictures and drawings), video clips (such as movies, newsreels, and home videos), audio clips (such as songs, phone messages, and speeches), and documents (such as books and articles). The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest. For example, one may want to locate all video clips in a video database that include a certain person. One may also want to retrieve video clips based on certain activities included in them, such as video clips where a soccer goal is scored by a certain player or team. The above types of queries are referred to as content-based retrieval, because the multimedia source is being retrieved based on its containing certain objects or activities. Hence, a multimedia database must use some model to organize and index the multimedia sources based on their contents. Identifying the contents of
333 multimedia sources is a difficult and time-consuming task. There are two main approaches. The first is based on automatic analysis of the multimedia sources to identify certain mathematical characteristics of their contents. This approach uses different techniques depending on the type of multimedia source (image, video, audio, or text). The second approach depends on manual identification of the objects and activities of interest in each multimedia source and on using this information to index the sources. This approach can be applied to all multimedia sources, but it requires a manual preprocessing phase in which a person must scan each multimedia source to identify and catalog the objects and activities it contains so that they can be used to index the sources. Storing multimedia outside the database makes it harder to provide database functionality, such as indexing on the basis of actual multimedia data content. It can also lead to inconsistencies, such as a file that is noted in the database, but whose contents are missing, or vice versa. It is therefore desirable to store the data themselves in the database. Several issues must be addressed if multimedia data are to be stored in a database: • The database must support large objects, since multimedia data such as videos can occupy up to a few gigabytes of storage. Many database systems do not support objects larger than a few gigabytes. Larger objects could be split into smaller pieces and stored in the database. Alternatively, the multimedia object may be stored in a file system, but the database may contain a pointer to the object; the pointer would typically be a file name. The SQL/MED standard (MED stands for Management of External Data) allows external data, such as files, to be treated as if they are part of the database. With SQL/MED, the object would appear to be part of the database, but can be stored externally. • The retrieval of some types of data, such as audio and video, has the requirement that data delivery must proceed at a guaranteed steady rate.
334 Such data are sometimes called isochronous data, or continuous-media data. For example, if audio data are not supplied in time, there will be gaps in the sound. If the data are supplied too fast, system buffers may overflow, resulting in loss of data. • Similarity-based retrieval is needed in many multimedia database applications. For example, in a database that stores fingerprint images, a query fingerprint image is provided, and fingerprints in the database that are similar to the query fingerprint must be retrieved. Index structures such as B+ trees and R-trees cannot be used for this purpose; special index structures need to be created. An image is typically stored either in raw form as a set of pixel or cell values, or in compressed form to save space. The image shape descriptor describes the geometric shape of the raw image, which is typically a rectangle of cells of a certain width and height. Hence, each image can be represented by an m by n grid of cells. Each cell contains a pixel value that describes the cell content. In blackand- white images, pixels can be one bit. In grayscale or color images, a pixel is multiple bits. Because images may require large amounts of space, they are often stored in compressed form. Compression standards, such as GIF, JPEG, or MPEG, use various mathematical transformations to reduce the number of cells stored but still maintain the main image characteristics. Applicable mathematical transforms include discrete Fourier transform (DFT), discrete cosine transform (DCT), and wavelet transforms. To identify objects of interest in an image, the image is typically divided into homogeneous segments using a homogeneity predicate. For example, in a color image, adjacent cells that have similar pixel values are grouped into a segment. The homogeneity predicate defines conditions for automatically grouping those cells. Segmentation and compression can hence identify the main characteristics of an image. A typical image database query would be to find images in the database that are

335 similar to a given image. The given image could be an isolated segment that contains, say, a pattern of interest, and the query is to locate other images that contain that same pattern. There are two main techniques for this type of search. The first approach uses a distance function to compare the given image with the stored images and their segments. If the distance value returned is small, the probability of a match is high. Indexes can be created to group stored images that are close in the distance metric so as to limit the search space. The second approach, called the transformation approach, measures image similarity by having a small number of transformations that can change one image's cells to match the other image. Transformations include rotations, translations, and scaling. Although the transformation approach is more general, it is also more time-consuming and difficult. A video source is typically represented as a sequence of frames, where each frame is a still image. However, rather than identifying the objects and activities in every individual frame, the video is divided into video segments, where each segment comprises a sequence of contiguous frames that includes the same objects/activities. Each segment is identified by its starting and ending frames. The objects and activities identified in each video segment can be used to index the segments. An indexing technique called frame segment trees has been proposed for video indexing. The index includes both objects, such as persons, houses, and cars, as well as activities, such as a person delivering a speech or two people talking. Videos are also often compressed using standards such as MPEG. Audio sources include stored recorded messages, such as speeches, class presentations, or even surveillance recordings of phone messages or conversations by law enforcement. Here, discrete transforms can be used to identify the main characteristics of a certain person's voice in order to have similarity-based indexing and retrieval. A text/document source is basically the full text of some article, book, or magazine. These sources are typically indexed by identifying the keywords that
336 appear in the text and their relative frequencies. However, filler words or common words called stopwords are eliminated from the process. Because there can be many keywords when attempting to index a collection of documents, techniques have been developed to reduce the number of keywords to those that are most relevant to the collection. A dimensionality reduction technique called singular value decomposition (SVD), which is based on matrix transformations, can be used for this purpose. An indexing technique called telescoping vector trees (TV-trees) can then be used to group similar documents. 16.3 Automatic Analysis of Images Analysis of multimedia sources is critical to support any type of query or search interface. We need to represent multimedia source data such as images in terms of features that would enable us to define similarity. The work done so far in this area uses low-level visual features such as color, texture, and shape, which are directly related to the perceptual aspects of image content. These features are easy to extract and represent, and it is convenient to design similarity measures based on their statistical properties. • Color is one of the most widely used visual features in content-based image retrieval since it does not depend upon image size or orientation. Retrieval based on color similarity is mainly done by computing a color histogram for each image that identifies the proportion of pixels within an image for the three color channels (red, green, blue—RGB). However, RGB representation is affected by the orientation of the object with respect to illumination and camera direction. Therefore, current image retrieval techniques compute color histograms using competing invariant representations such as HSV (hue, saturation, value). HSV describes colors as points in a cylinder whose central axis ranges from black at the bottom to white at the top with neutral colors between them. The angle around the axis corresponds to the hue, the distance from the axis corresponds to the saturation, and the distance along
337 the axis corresponds to the value (brightness). • Texture refers to the patterns in an image that present the properties of homogeneity that do not result from the presence of a single color or intensity value. Examples of texture classes are rough and silky. Examples of textures that can be identified include pressed calf leather, straw matting, cotton canvas, and so on. Just as pictures are represented by arrays of pixels (picture elements), textures are represented by arrays of texels (texture elements). These textures are then placed into a number of sets, depending on how many textures are identified in the image. These sets not only contain the texture definition but also indicate where in the image the texture is located. Texture identification is primarily done by modeling it as a two-dimensional, gray-level variation. The relative brightness of pairs of pixels is computed to estimate the degree of contrast, regularity, coarseness, and directionality. • Shape refers to the shape of a region within an image. It is generally determined by applying segmentation or edge detection to an image. Segmentation is a region-based approach that uses an entire region (sets of pixels), whereas edge detection is a boundary-based approach that uses only the outer boundary characteristics of entities. Shape representation is typically required to be invariant to translation, rotation, and scaling. Some well-known methods for shape representation include Fourier descriptors and moment invariants. 16.4 Object Recognition in Images Object recognition is the task of identifying real-world objects in an image or a video sequence. The system must be able to identify the object even when the images of the object vary in viewpoints, size, scale, or even when they are rotated or translated. Some approaches have been developed to divide the original image into regions based on similarity of contiguous pixels. Thus, in a given image

338 showing a tiger in the jungle, a tiger subimage may be detected against the background of the jungle, and when compared with a set of training images, it may be tagged as a tiger. The representation of the multimedia object in an object model is extremely important. • One approach is to divide the image into homogeneous segments using a homogeneous predicate. For example, in a colored image, adjacent cells that have similar pixel values are grouped into a segment. The homogeneity predicate defines conditions for automatically grouping those cells. Segmentation and compression can hence identify the main characteristics of an image. • Another approach finds measurements of the object that are invariant to transformations. It is impossible to keep a database of examples of all the different transformations of an image. To deal with this, object recognition approaches find interesting points (or features) in an image that are invariant to transformations. An important contribution to this field was made by Lowe, who used scale-invariant features from images to perform reliable object recognition. This approach is called scale-invariant feature transform (SIFT). The SIFT features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition. For image matching and recognition, SIFT features (also known as keypoint features) are first extracted from a set of reference images and stored in a database. Object recognition is then performed by comparing each feature from the new image with the features stored in the database and finding candidate
339 matching features based on the Euclidean distance of their feature vectors. Since the keypoint features are highly distinctive, a single feature can be correctly matched with good probability in a large database of features. In addition to SIFT, there are a number of competing methods available for object recognition under clutter or partial occlusion. For example, RIFT, a rotation invariant generalization of SIFT, identifies groups of local affine regions (image features having a characteristic appearance and elliptical shape) that remain approximately affinely rigid across a range of views of an object, and across multiple instances of the same object class. 16.5 Semantic Tagging of Images The notion of implicit tagging is an important one for image recognition and comparison. Multiple tags may attach to an image or a subimage: for instance, in the example we referred to above, tags such as “tiger,” “jungle,” “green,” and “stripes” may be associated with that image. Most image search techniques retrieve images based on user-supplied tags that are often not very accurate or comprehensive. To improve search quality, a number of recent systems aim at automated generation of these image tags. In case of multimedia data, most of its semantics is present in its content. These systems use image-processing and statistical-modeling techniques to analyze image content to generate accurate annotation tags that can then be used to retrieve images by content. Since different annotation schemes will use different vocabularies to annotate images, the quality of image retrieval will be poor. To solve this problem, recent research techniques have proposed the use of concept hierarchies, taxonomies, or ontologies using OWL (Web Ontology Language), in which terms and their relationships are clearly defined. These can be used to infer higher-level concepts based on tags. Concepts like “sky” and “grass” may be further divided into “clear sky” and “cloudy sky” or “dry grass” and “green grass” in such a taxonomy. These approaches generally come under semantic tagging and can be used in
340 conjunction with the above feature-analysis and object-identification strategies. 16.6 Analysis of Audio Data Sources Audio sources are broadly classified into speech, music, and other audio data. Each of these is significantly different from the others; hence different types of audio data are treated differently. Audio data must be digitized before it can be processed and stored. Indexing and retrieval of audio data is arguably the toughest among all types of media, because like video, it is continuous in time and does not have easily measurable characteristics such as text. Clarity of sound recordings is easy to perceive humanly but is hard to quantify for machine learning. Interestingly, speech data often uses speech recognition techniques to aid the actual audio content, as this can make indexing this data a lot easier and more accurate. This is sometimes referred to as text-based indexing of audio data. The speech metadata is typically content dependent, in that the metadata is generated from the audio content; for example, the length of the speech, the number of speakers, and so on. However, some of the metadata might be independent of the actual content, such as the length of the speech and the format in which the data is stored. Music indexing, on the other hand, is done based on the statistical analysis of the audio signal, also known as content- based indexing. Content-based indexing often makes use of the key features of sound: intensity, pitch, timbre, and rhythm. It is possible to compare different pieces of audio data and retrieve information from them based on the calculation of certain features, as well as application of certain transforms. 16.7
Introduction to Mobile Databases We are currently witnessing increasing demands on mobile computing to provide the types of support required by a growing number of mobile workers
and end customers; for example, mobile customer relationship management (CRM)/sales force automation (SFA). Such individuals must be able
to work as if in the office,
341

but in reality are working from remote locations including homes, clients' premises, or simply while en route to remote locations. The "office" may accompany a remote worker in the form of a laptop, smartphone, tablets, or other Internet access device. With the rapid expansion of cellular, wireless, and satellite communications, it
is
possible for mobile users to access any data, anywhere, at any time.
According to Cisco's Global Mobile Data Traffic Forecast (Cisco, 2012), global mobile data traffic will increase 18-fold between 2011 and 2016. Mobile data traffic will grow at a compound annual growth rate (CAGR) of 78% from 2011 to 2016, reaching 10.8 exabytes (1018 bytes) per month by 2016. By the end of 2012, the number of mobile connected devices will exceed the number of people on earth, and by 2016 there will be 1.4 mobile devices per capita. There will be over 10 billion mobile-connected devices in 2016, including machine-to- machine (M2M) modules exceeding the world's population at that time (7.3 billion). However,
business etiquette, practicalities, security, and costs may still limit communication such that it is not possible to establish online connections for as long as users want, whenever they want. Mobile databases offer a solution for some of these restrictions. Mobile
database
is a database that is portable and physically separate from the corporate database server, but is capable of communicating with that server from remote sites allowing the sharing of corporate data.
With
mobile databases, users have access to corporate data on their laptop, smartphone, or other Internet access device that is required for applications at remote sites. The typical architecture for a mobile database environment is shown in Figure 16.1. The components of a mobile database environment include: • Corporate database server and DBMS that manages and stores the corporate data and provides corporate applications; • Remote database and DBMS that manages and stores the mobile data and provides mobile applications;
342 • Mobile database platform that includes laptop, smartphone, or other Internet access devices; • Two-way communication links between the corporate and mobile DBMS.
Figure 16.1 Typical architecture for a mobile database environment.
Depending on the particular requirements of mobile applications, in some cases the user of a mobile device may log on to a corporate database server and work
with data there; in others the user may download data and work with it on a mobile device or upload data captured at the remote site to the corporate database. The communication between the corporate and mobile databases is usually intermittent and is typically established for short periods of time at irregular intervals. Although unusual, there are some applications that require direct communication between the mobile databases. The two main issues associated with mobile databases are the management of the mobile database and the communication between the mobile and corporate databases. In the following
343 section we identify the requirements of mobile DBMSs. All the major DBMS vendors now offer mobile DBMS or middleware solutions enabling the access to their DBMS solutions.
In fact, this development is partly responsible for driving the current dramatic growth in sales for the major DBMS vendors. Most vendors promote their mobile DBMS as being capable of communicating with a range of major relational DBMSs and in providing database services that require limited computing resources to match those currently provided by mobile devices. The additional functionality required of mobile DBMSs includes the ability to: • communicate with the centralized database server through modes such as wireless or Internet access; • replicate data on the centralized database server and mobile device; • synchronize data on the centralized database server and mobile device; • capture data from various sources such as the Internet; • manage data on the mobile device; • analyze data on a mobile device; • create customized mobile applications. DBMS vendors are driving the prices per user to such a level that it is now cost effective for organizations to extend applications to mobile devices, where the applications were previously available only in-house. Currently, most mobile DBMSs only provide prepackaged SQL functions for the mobile application, rather than supporting any extensive database querying or data analysis. However, the prediction is that in the near future mobile devices will offer functionality that at least matches the functionality available at the corporate site. 16.8
Summary • Multimedia databases are growing in importance. Issues such as similarity

344 based retrieval and delivery of data at guaranteed rates are topics of current research. • The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest. • An image is typically stored either in raw form as a set of pixel or cell values, or in compressed form to save space. • A video source is typically represented as a sequence of frames, where each frame is a still image. • Audio sources include stored recorded messages, such as speeches, class presentations, or even surveillance recordings of phone messages or conversations by law enforcement. • A text/document source is basically the full text of some article, book, or magazine. These sources are typically indexed by identifying the keywords that appear in the text and their relative frequencies. •
Mobile database is a database that is portable and physically separate from the corporate database server, but is capable of communicating with that server from remote sites allowing the sharing of corporate data. 16.9
Key Terms • Multimedia databases: Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images, video clips, audio clips , and documents. • Isochronous data: The retrieval of some types of data, such as audio and video, has the requirement that data delivery must proceed at a guaranteed steady rate. Such data are sometimes called isochronous data, or continuous-media data. • Pixel: It is the smallest unit of an image.
345 • Segmentation: It is a region-based approach that uses an entire region (sets of pixels). 16.10 Check Your Progress Short- Answer Type Q1) An image is typically stored either in raw form as a set of _____ or cell values. Q2) Full form of SIFT- a) scale-invariant feature transform b) size-invariant feature transform c) scale-invalid feature test d) size-invalid feature test Q3) Shape representation is typically required to be invariant to translation, rotation, and scaling. (True/ False?) Q4) _____ refers to the patterns in an image that present the properties of homogeneity that do not result from the presence of a single color or intensity value. Q5) Full form of SVD- a) singular value decomposition b) simple value decrement c) singular value decrement d) simple value decomposition Long- Answer Type Q1) What is a mobile database? Explain the basic architecture of a mobile database environment. Q2) What is a content-based retrieval? Give an example. Q3) Write a short note on Semantic tagging of Images. Q4) What are the different approaches to recognizing objects in images?
346 Q5) What are the difficulties in analyzing audio sources? References • Fundamentals of Database Systems, R. Elmasri, S.B. Navathe, Fifth Edition, Pearson Education/Addison Wesley, 2007. • Database System Concepts, Henry F Korth, Abraham Silberschatz, and S. Sudharshan, 6th Edition, McGraw Hill, 2011.
347
APPENDIX- I: ARIES Algorithms for Recovery and Isolation Exploiting Semantics, or ARIES is a recovery algorithm designed to work with a no-force, steal database approach; it is used by IBM DB2, Microsoft SQL Server and many other database systems. The state of the art in recovery methods is best illustrated by the ARIES recovery method. The recovery technique is modeled after ARIES, but has been simplified significantly to bring out key concepts and make it easier to understand. In contrast,

100% **MATCHING BLOCK 196/202** **W**

ARIES uses a number of techniques to reduce the time taken for recovery, and to reduce the overhead of checkpointing. In particular, ARIES is able to avoid redoing many logged operations that have already been applied and to reduce the amount of information logged. The price paid is greater complexity; the benefits are worth the price. The major differences between ARIES and the recovery algorithm presented earlier are that ARIES: 1. Uses a log sequence number (LSN) to identify log records, and stores LSNs in database pages to identify which operations have been applied to a database page. 2. Supports physiological redo operations, which are physical in that the affected page is physically identified, but can be logical within the page.

For instance, the deletion of a record from a page may result in many other records in the page being shifted, if a slotted page structure is used. With physical redo logging, all bytes of the page affected by the shifting of records must be logged.With physiological logging, the deletion operation can be logged, resulting in a much smaller log record. Redo of the deletion operation would delete the record and shift other records as required. 3.

100% **MATCHING BLOCK 197/202** **W**

Uses a dirty page table to minimize unnecessary redos during recovery. As mentioned earlier, dirty pages are those that have been updated in memory, and the disk version is not up-to-date. 348 4. Uses a fuzzy-checkpointing scheme that records only information about dirty pages and associated information and does not even require writing of dirty pages to disk. It flushes dirty pages in the background, continuously, instead of writing them during checkpoints.

Data Structures in ARIES Each log record in ARIES has a log sequence number (LSN) that uniquely identifies the record. The number is conceptually just a logical identifier whose value is greater for log records that occur later in the log. In practice, the LSN is generated in such a way that it can also be used to locate the log record on disk. Typically, ARIES splits a log into multiple log files, each of which has a file number. When a log file grows to some limit, ARIES appends further log records to a new log file; the new log file has a file number that is higher by 1 than the previous log file. The LSN then consists of a file number and an offset within the file. Each page also maintains an identifier called the PageLSN. Whenever an update operation (whether physical or physiological) occurs on a page, the operation stores the LSN of its log record in the PageLSN field of the page. During the redo phase of recovery, any 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 on the page. In combination with a scheme for recording PageLSNs as part of checkpointing, which we present later, ARIES can avoid even reading many pages for which logged operations are already reflected on disk. Thereby, recovery time is reduced significantly. The PageLSN is essential for ensuring idempotence in the presence of physiological redo operations, since reapplying a physiological redo that has already been applied to a page could cause incorrect changes to a page. Pages should not be flushed to disk while an update is in progress, since physiological operations cannot be redone on the partially updated state of the page on disk.
349 Therefore, ARIES uses latches on buffer pages to prevent them from being written to disk while they are being updated. It releases the buffer page latch only after the update is completed, and the log record for the update has been written to the log. Each log record also contains the LSN of the previous log record of the same transaction. This value, stored in the PrevLSN field, permits log records of a transaction to be fetched backward, without reading the whole log. There are special redo-only log records generated during transaction rollback, called compensation log records (CLRs) in ARIES. These serve the same purpose as the redo-only log records in our earlier recovery scheme. In addition CLRs serve the role of the operation-abort log records in our scheme. The CLRs have an extra field, called the UndoNextLSN, that records the LSN of the log that needs to be undone next, when the transaction is being rolled back. This field serves the same purpose as the operation identifier in the operation-abort log record in our earlier recovery scheme, which helps to skip over log records that have already been rolled back.

98% **MATCHING BLOCK 198/202** W

Recovery Algorithm ARIES recovers from a system crash in three passes: • Analysis pass: This pass determines which transactions to undo,which pages were dirty at the time of the crash, and the LSN from which the redo pass should start. • Redo pass: This pass starts from a position determined during analysis, and performs a redo, repeating history, to bring the database to a state it was in before the crash. • Undo pass: This pass rollback all transactions that were incomplete at the time of crash. 350 • Other Features Among other key features that ARIES provides are: • Nested top actions: ARIES allows the logging of operations that should not be undone even if a transaction gets rolled back;

for example, if a transaction allocates a page to a relation, even if the transaction is rolled back the page allocation should not be undone since other transactions may have stored records in the page.

100% **MATCHING BLOCK 199/202** W

Such operations that should not be undone are called nested top actions.

Such operations can be modeled as operations whose undo action does nothing. In ARIES, such operations are implemented by creating a dummy CLR whose UndoNextLSN is set such that transaction rollback skips the log records generated by the operation. •

100% **MATCHING BLOCK 200/202** W

Recovery independence: Some pages can be recovered independently from others, so that they can be used even while other pages are being recovered. If some pages of a disk fail, they can be recovered without stopping transaction processing on other pages. • Savepoints: Transactions can record savepoints, and can be rolled back partially, up to a savepoint. This can be quite useful for deadlock handling, since transactions can be rolled back up to a point that permits release of required locks, and then restarted from that point. Programmers can also use savepoints to undo a transaction partially, and then continue execution;

this approach can be useful to handle certain kinds of errors detected during the transaction

execution. • Fine-grained locking: The ARIES recovery algorithm can be used with index concurrency-control algorithms that permit tuple-level locking on indices, instead of page-level locking, which improves concurrency significantly. • Recovery optimizations: The DirtyPageTable can be used to prefetch pages during redo, instead of fetching a page only when the system finds a log record to be applied to the page. Out-of-order redo is also possible: Redo 351 can be postponed on a page being fetched from disk, and performed when the page is fetched. Meanwhile, other log records can continue to be processed. •

In summary, the ARIES algorithm is a state-of-the-art recovery algorithm, incorporating a variety of optimizations designed to improve concurrency, reduce logging overhead, and reduce recovery time.
352 APPENDIX-II: ORACLE When Oracle was founded in 1977 as Software Development Laboratories by Larry Ellison, Bob Miner, and Ed Oates, there were no commercial relational database products. The company, which was later renamed Oracle, set out to build a relational database management system as a commercial product, and became a pioneer of the RDBMS market and has held a leading position in this market ever since. Over the years, its product and service offerings have grown beyond the relational database server to include middleware and applications. In addition to products developed inside the company, Oracle's offerings include software that was originally developed in companies that Oracle acquired. Oracle's acquisitions have ranged from small companies to large, publicly traded ones, including Peoplesoft, Siebel, Hyperion, and BEA. As a result of these acquisitions, Oracle has a very broad portfolio of enterprise software products. Here we will focus on Oracle's main relational database server and closely related products. New versions of the products are being developed continually, so all product descriptions are subject to change. The feature set described here is based on the first release of Oracle11g, which is Oracle's flagship database product. Database Design and Querying Tools Oracle provides
a variety of tools for database design, querying, report generation and data analysis,
including OLAP. These tools, along with various other application development tools, are part of a portfolio of software products called Oracle Fusion Middleware. Products include both traditional tools using Oracle's PL/SQL programming language and newer ones based on Java/J2EE technologies. The software supports open standards such as SOAP, XML, BPEL,
353 and UML. The Oracle Application Development Framework (ADF) is an end-to-end J2EEbased development framework for a Model-View-Control design pattern. In this framework, an application consists of multiple layers. The Model and Business Services layers handle the interaction with the data sources and contain the business logic. The Viewlayer handles the user interface, and the Controller layer handles the flow of the application and the interaction between the other layers. The primary development tool for Oracle ADF is Oracle JDeveloper, which provides an integrated development environment with support for Java, XML, PHP, HTML, Javascript, BPEL, SQL, andPL/SQL development. It has built-in support for UML modeling.
Oracle Designer is a database design tool,which translates business logic and data flows into schema definitions and procedural scripts for application logic. It supports such modeling techniques as E-R diagrams, information engineering, and object analysis and design.
Oracle also has an application development tool for data warehousing, Oracle Warehouse Builder. Warehouse Builder is a
tool for design and deployment of all aspects of a data warehouse, including schema design, data mapping and transformations, data load processing, and metadata management.
Oracle Warehouse Builder
supports both 3NF and star schemas and can also import designs from Oracle Designer.
This tool, in conjunction with database features, such as external tables and table functions, typically eliminates the need for
third-party extraction, transformation, and loading (ETL) tools. Oracle provides tools for ad hoc querying, report generation, and data analysis, including OLAP.
Oracle Business Intelligence Suite (OBI) is a comprehensive suite of tools sharing a common service-oriented architecture. Components include a Business Intelligence server and tools for ad hoc querying, dashboard generation, reporting, and alerting. The components share infrastructure and services for data access and metadata management and have a common security model and

354 administration tool. The component for ad hoc querying, Oracle BI Answers, is an interactive tool that presents the user with a logical view of the data hiding the details of the physical implementation. Objects available to the user are displayed graphically and the user can build a query with a point-and-click interface. This logical query is sent to the Oracle BI Server component,which then generates the physical query or queries. Multiple physical data sources are supported, and a query could combine data stored in relational databases, OLAP sources, and Excel spreadsheets. Results can be presented as charts, reports, pivot tables, or dashboards that are drillable and can be saved and later modified.
SQL Variations and Extensions Oracle supports all core SQL:2003 features fully or partially, with the exception of features-and-conformance views.
In addition, Oracle supports a large number of other language constructs, some of which conform with Optional Features of SQL Foundation:2003, while others are Oracle-specific in syntax or functionality.
Oracle has extensive support for object-relational constructs, including: • Object types.
A single-inheritance model is supported for type hierarchies. • Collection types. Oracle supports varrays, which are variable length arrays, and nested tables. • Object tables. These are used to store objects while providing a relational view of the attributes of the objects. • Table functions. These are functions that produce sets of rows as output, and can be used in the from clause of a query. Table functions in Oracle can be nested. If a table function is used to express some form of data transformation, nesting multiple functions allows multiple transformations to be expressed in a single statement. • Object views. These provide a virtual object table view of data stored in a regular relational table. They allow data to be accessed or viewed in an object oriented style even if the data are really stored in a traditional
355 relational format. • Methods. These can be written in PL/SQL, Java, or C. • User-defined aggregate functions. These can be used in SQL statements in the same way as built-in functions such as sum and count. Oracle XML DB Oracle XML DB provides in-database storage for XML data and support for a broad set of XML functionality including XML Schema and XQuery. It is built on the XMLType abstract data type, which is treated as a native Oracle data type. XML DB provides several options for how data of this data type are stored in the database, including: • Structured in object-relational format. This format is usually space efficient and allows the use of a variety of standard relational features, such as B-tree indices, but incurs some overhead when mapping XML documents to the storage format and back. It is mainly suitable for XML data that are highly structured and the mapping includes a manageable number of relational tables and joins. • Unstructured as a text string. This representation does not require any mapping and provides high throughput when inserting or retrieving an entire XML document. However, it is usually not very space efficient and provides for less intelligent processing when operating on parts of an XML document. • Binary XML storage. This representation is a post-parse, XML Schema- aware, binary format. It is more space efficient than unstructured storage and can handle operations against parts of an XML document. It is also better than the structured format at handling data that are highly unstructured, but may not always be as space efficient. This format may make the processing of XQuery statements less efficient than when the structured format is used.
356 Both the binary and unstructured representation can be indexed with a special type of index called XMLIndex. This type of index allows document fragments to be indexed based on their corresponding XPath expression. Storing XML data inside the database means that they get the benefit of Oracle's functionality in areas such as backup, recovery, security, and query processing. It allows for accessing relational data as part of doing XML processing as well as accessing XML data as part of doing SQL processing. Some very high-level features of XML DB include: • Support for the XQuery language (W3C XQuery 1.0 Recommendation). • An XSLT process that lets XSL transformations be performed inside the database. • An XPath rewrite optimization that can speed up queries against data stored in object-relational representation. By translating an expression used in an XQuery into conditions directly on the object-relational columns, regular indices on these columns can be used to speed up query processing. Procedural Languages Oracle has two main procedural languages, PL/SQL and Java. PL/SQL was Oracle's original language for stored procedures and it has syntax similar to that used in the Ada language. Java is supported through a Java virtual machine inside the database engine. Oracle provides a package to encapsulate related procedures, functions, and variables into single units. Oracle supports SQLJ (SQL embedded in Java) and JDBC, and provides a tool to generate Java class definitions corresponding to user-defined database types. OLAP Oracle provides support for analytical database processing in several different ways. In addition to support for a rich set of analytical SQL constructs (cube, rollup, grouping sets, window functions, etc.), Oracle provides native

357 multidimensional storage inside the relational database server. The multidimensional data structures allow for array-based access to the data, and, in the right circumstances, this type of access can be vastly more efficient than traditional relational access methods. Using these data structures as an integrated part of a relational database provides a choice of storing data in a relational or multidimensional format while still taking advantage of Oracle features in areas such as backup and recovery, security, and administration tools. Oracle provides storage containers for multidimensional data known as analytic workspaces. An analytic workspace contains both the dimensional data and measures (or facts) of an OLAP cube and is stored inside an Oracle table. From a traditional relational perspective, a cube stored inside a table would be an opaque object where the data could not normally be interpreted directly in terms of the table's rows and columns. However, Oracle's OLAP server inside the database has the knowledge to interpret and access the data and makes it possible to give SQL access to it as if it had been stored in a regular table format. Hence, it is possible to store data in either a multidimensional format or a traditional relational format, depending on what is optimal, and still be able to join data stored in both types of representations in a single SQLquery. A materialized view can use either representation. In addition to Oracle's OLAP support inside its relational database, Oracle's product suite includes Essbase. Essbase is a widely used multidimensional database that came to be part of Oracle with the acquisition of Hyperion. Storage and Indexing In Oracle parlance, a database consists of information stored in files and is accessed through an instance,which is a shared memory area and
a
set of processes that interact with the data in the files.
The control file is a small file that contains some very high-level metadata required to start or operate an
358 instance. A database consists of one or more logical storage units called table spaces. Each table space, in turn, consists of one or more physical structures called data files. These may be either part of a file system or raw devices. Usually, an Oracle database has the following table spaces: • The system and the auxiliary sysaux table spaces, which are always created. They contain the data dictionary tables and storage for triggers and stored procedures. • Table spaces created to store user data. While user data can be stored in the system tablespace, it is often desirable to separate the user data from the system data. Usually, the decision about what other table spaces should be created is based on performance, availability, maintainability, and ease of administration. For example, having multiple tablespaces can be useful for partial backup and recovery operations. • The undo tablespace, which is used solely for storing undo information for transaction management and recovery. • Temporary table spaces. Many database operations require sorting the data, and the sort routine may have to store data temporarily on disk if the sort cannot be done in memory. Temporary table spaces are allocated for sorting and hashing to make the space management operations involved in spilling to disk more efficient. The space in a table space is divided into units, called segments, each of which contains data for a specific data structure. There are four types of segments: • Data segments. Each table in a table space has its own data segment where the table data is stored unless the table is partitioned; if so, there is one data segment per partition. • Index segments. Each index in a table space has its own index segment,
359 except for partitioned indices, which have one index segment per partition. • Temporary segments. These are segments used when a sort operation needs to write data to disk or when data is inserted into a temporary table. • Undo segments. These segments contain undo information so that an uncommitted transaction can be rolled back. These segments are automatically allocated in a special undo table space. They also play an important role in Oracle's concurrency control model and for database recovery. Below the level of segment, space is allocated at a level of granularity called an extent. Each extent consists of a set of contiguous database blocks. A database block is the lowest level of granularity at which Oracle performs disk I/O.
A database block does not have to be the same as an operating system block in size, but should be a multiple thereof. • Oracle provides storage parameters that allow for detailed control of how space is allocated and managed, parameters such as: • The size of a new extent that is to be allocated to provide room for rows that are inserted into a table. • The percentage of space utilization at which a database block is considered full and at which no more rows will be inserted into that block. (Leaving some free space in a block can allow the existing rows to grow in size through updates, without running out of space in the block.) Data Security In addition to regular access control features such as passwords, user privileges, and user roles, Oracle supports several features to protect the data from unauthorized access, including: • Encryption. Oracle can automatically store table data in an encrypted

360 format and transparently encrypt and decrypt data using the AES or 3DES algorithms. Encryption can be enabled for an entire database or just for individual table columns. The main motivation for this feature is to protect sensitive data outside the normally protected environment, such as when backup media is sent to a remote location. • Database Vault. This feature is aimed at providing a separation of duties for users with access to the database. A database administrator is a highly privileged user that typically can do almost anything with the database.However, it may be inappropriate or illegal to let that person access sensitive corporate financial data or personal information about other employees. Database vault includes a variety of mechanisms that can be used to restrict or monitor access to sensitive data by highly privileged database users. • Virtual Private Database. This feature allows additional predicates to be automatically added to the where clause of a query that accesses a given table or view. Typically, the feature would be used so that the additional predicate filters out all the rows that the user does not have the right to see. For example, two users could submit identical queries to find all the employee information in the entire employee table. However, if a policy exists that limits each user to seeing only the information for the employee number that matches the user ID, the automatically added predicate will ensure that each query only returns the employee information for the user who submitted the query. Hence, each user will be left with the impression of accessing a virtual database that contains only a subset of the data of the physical database. Partitioning Oracle supports various kinds of horizontal partitioning of tables and indices, and this feature plays a major role in Oracle's ability to support very large databases.
361 The ability to partition a table or index has advantages in many areas. • Backup and recovery are easier and faster, since they can be done on individual partitions rather than on the table as a whole. • Loading operations in a data warehousing environment are less intrusive: data can be added to a newly created partition, and then the partition added to a table,which is an instantaneous operation. Likewise, dropping a partition with obsolete data from a table is very easy in a data warehouse that maintains a rolling window of historical data. • Query performance benefits substantially, since the optimizer can recognize that only a subset of the partitions of a table need to be accessed in order to resolve a query (partition pruning). Also, the optimizer can recognize that in a join, it is not necessary to try to match all rows in one table with all rows in the other, but that the joins need to be done only between matching pairs of partitions (partition wise join). Concurrency Control and Recovery Oracle supports concurrency control and recovery techniques that provide a number of useful features. Oracle's multiversion concurrency control mechanism is based on the snapshot isolation protocol. Read-only queries are given a read consistent snapshot, which is a view of the database as it existed at a specific point in time, containing all updates that were committed by that point in time, and
not
containing any updates that were not committed at that point in time. Thus, read locks are not used and read-only queries do not interfere with other database activity in terms of locking. Oracle supports both statement- and transaction-level read consistency:
at the beginning of the execution of either a statement or a transaction (depending on what level of consistency is used), Oracle determines the current system change number (SCN). The SCN essentially acts as a timestamp, where the time is measured in terms of transaction commits instead of wall-clock time. If in the course of a query a data block is found that has a higher SCN than the one being
362 associated with the query, it is evident that the data block has been modified after the time of the original query's SCN by some other transaction that may or may not have committed. Hence, the data in the block cannot be included in a consistent view of the database as it existed at the time of the query's SCN. Instead, an older version of the data in the block must be used; specifically, the one that has the highest SCN that does not exceed the SCN of the query. Oracle retrieves that version of the data from the undo segment. Hence, provided that the undo space is sufficiently large, Oracle can return a consistent result of the query even if the data items have been modified several times since the query started execution. Should the block with the desired SCN no longer exist in the undo, the query will return an error. It would be an indication that the undo tablespace has not been properly sized, given the activity on the system.
In the Oracle concurrency model, read operations do not block write operations and write operations do not block read operations,

a property that allows a high degree of concurrency. In particular, the scheme allows for long-running queries (for example, reporting queries) to run on a system with a large amount of transactional activity. This kind of scenario is often problematic for database systems where queries use read locks, since the query may either fail to acquire them or lock large amounts of data for a long time, thereby preventing transactional activity against that data and reducing concurrency. (An alternative that is used in some systems is to use a lower degree of consistency, such as degree-two consistency, but that could result in inconsistent query results.) Oracle's concurrency model is used as a basis for the flashback feature. This feature allows a user to set a certain SCN number or wall-clock time in his session and perform operations on the data that existed at that point in time (provided that the data still exists in the undo). Normally in a database system, once a change has been committed, there is no way to get back to the previous state of the data other than performing point-in-time recovery from backups. However, recovery of a very large database can be very costly, especially if the goal is just to retrieve some data item that had been inadvertently deleted by a user.
363 The flashback feature provides a much simpler mechanism to deal with user errors. The flashback feature includes the ability to restore a table or an entire database to an earlier point in time without recovering from backups, the ability to perform queries on the data as they existed at an earlier point in time, the ability to track how one or more rows have changed over time, and the ability to examine changes to the database at the transaction level.
Oracle detects deadlocks automatically and resolves them by rolling back one of the transactions involved in the deadlock.
Oracle supports autonomous transactions, which are independent transactions generated within other transactions. When Oracle invokes an autonomous transaction, it generates a new transaction in a separate context. The new transaction can be either committed or rolled back before control returns to the calling transaction. Oracle supports multiple levels of nesting of autonomous transactions. Oracle Data Guard To ensure high availability, Oracle provides a standby database feature, data guard. (This feature is the same as remote backups) A standby database is a copy of the regular database that is installed on a separate system. If a catastrophic failure occurs on the primary system, the standby system is activated and takes over, thereby minimizing the effect of the failure on availability. Oracle keeps the standby database up-to-date by constantly applying archived redo logs that are shipped from the primary database. The backup database can be brought online in read-only mode and used for reporting and decision support queries. Replication, Distribution, and External Data Oracle provides support for replication and distributed transactions with two phase commit protocol. Oracle supports several types of replication. In one form, data in a master site are replicated to other sites in the form of materialized views. A materialized view does not have to contain all the master data—it can, for example, exclude certain columns from a table for security reasons. Oracle
364 supports two types of materialized views for replication: read-only and updatable. An updatable materialized view can be modified and the modifications propagated to the master table. However, read-only materialized views allow for a wider range of view definitions. For instance, a read-only materialized view can be defined in terms of set operations on tables at the master site. Changes to the master data are propagated to the replicas through the materialized view refresh mechanism. Oracle supports queries and transactions spanning multiple databases on different systems.With the use of gateways, the remote systems can include non- Oracle databases. Oracle has built-in capability to optimize a query that includes tables at different sites, retrieve the relevant data, and return the result as if it had been a normal, local query. Oracle also transparently supports transactions spanning multiple sites by a built-in two-phase-commit protocol. Oracle has several mechanisms for supporting external data sources. The most common usage is in data warehousing when large amounts of data are regularly loaded from a transactional system.

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.

1/202	**SUBMITTED TEXT**	29 WORDS	**64%**	**MATCHING TEXT**	29 WORDS
Databases and database systems have become an essential component of a modern lifestyle. Most of us encounter several activities every day that involve some interaction with a database.			Databases and database systems have become an essential component of everyday life in modern society. In the course of a day, most of us encounter several activities that involve some interaction with a database.		

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

2/202	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

activities will involve someone or some computer program accessing a database.

activities will involve someone or some computer program accessing a database.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

3/202	SUBMITTED TEXT	18 WORDS	88%	MATCHING TEXT	18 WORDS

In the past few years, advances in technology have led to exciting new applications of database systems.

In the past few years, advances in technology have been leading to exciting new applications of database systems.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

4/202	SUBMITTED TEXT	37 WORDS	93%	MATCHING TEXT	37 WORDS

One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files,

One way to keep the information on a computer is to store it in permanent system files. To allow users to manipulate the stored information, the system has a number of application programs that manipulate the organized files.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

5/202	SUBMITTED TEXT	22 WORDS	83%	MATCHING TEXT	22 WORDS

bank deposit or withdraw funds, a hotel or airline reservation, a computerized library catalog to search for a bibliographic item, or

bank to deposit or withdraw funds, if we make a hotel or airline reservation, if we access a computerized library catalog to search for a bibliographic item, or

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

6/202	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

New application programs are added to the system as the need arises.

New application programs are added to the system as the need arises.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

7/202	SUBMITTED TEXT	47 WORDS	83%	MATCHING TEXT	47 WORDS

information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including

SA 47F417BA15858476660.pdf (D123781188)

8/202	SUBMITTED TEXT	17 WORDS	100%	MATCHING TEXT	17 WORDS

New application programs are added to the system as the need arises. For example, suppose that

SA 47F417BA15858476660.pdf (D123781188)

9/202	SUBMITTED TEXT	58 WORDS	68%	MATCHING TEXT	58 WORDS

in a file-processing system has a number of major disadvantages: • Data redundancy and inconsistency Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (

in a file processing system has a number of disadvantages, namely FILE MANAGEMENT SYSTEM PROBLEMS • Data Redundancy and Inconsistency: Since the files and applications programs are created by different programmers over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

10/202	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

higher storage and access cost. In addition, it may lead to data inconsistency;

higher storage and access cost. In addition, it may lead to data inconsistency. •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

11/202	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

may have to write new application programs to deal with

SA 47F417BA15858476660.pdf (D123781188)

12/202	SUBMITTED TEXT	17 WORDS	82%	MATCHING TEXT	17 WORDS

file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

file processing system does not allow needed data to be retrieved in a convenient and efficient manner. •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

13/202	SUBMITTED TEXT	23 WORDS	75%	MATCHING TEXT	23 WORDS

data is scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data

data are scattered in various files, and files may be in different formats. It is very difficult to write new application programs to retrieve the appropriate data. •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

14/202	SUBMITTED TEXT	29 WORDS	91%	MATCHING TEXT	29 WORDS

Integrity problems The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below

Integrity problems: The data values stored in the database must satisfy certain types of consistency Constraints (For example, the minimum balance in a bank account may never fall below

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

15/202	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

Developers enforce these constraints in the system by adding appropriate code in the

Developers enforce these constraints in the system by adding appropriate code in the

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

16/202	SUBMITTED TEXT	19 WORDS	97%	MATCHING TEXT	19 WORDS

application programs. However, when new constraints are added, it is difficult to change the programs to enforce them.

application programs. However, when new constraints are added, it is difficult to change the application programs to enforce them. •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

17/202	SUBMITTED TEXT	214 WORDS	98%	MATCHING TEXT	214 WORDS

Thus, as time goes by, the system acquires more files and more application programs. This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in a file-processing system has a number of major disadvantages: • Data redundancy and inconsistency Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system. Difficulty in accessing data. 7 •

SA 47F417BA15858476660.pdf (D123781188)

18/202	SUBMITTED TEXT	132 WORDS	91%	MATCHING TEXT	132 WORDS

device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system. • Concurrent-access anomalies For the sake of

device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer Rs.500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the Rs.500 was removed from the balance of department A but was not credited to the balance of department B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic — it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system. • Concurrent-access anomalies. For the sake of

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

19/202	SUBMITTED TEXT	19 WORDS	100%	MATCHING TEXT	19 WORDS

overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.

overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

20/202	SUBMITTED TEXT	15 WORDS	100%	MATCHING TEXT	15 WORDS

A computer system, like any other mechanical or electrical device, is subject to failure.

A computer system, like any other mechanical or electrical device, is subject to failure.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

21/202	SUBMITTED TEXT	39 WORDS	100%	MATCHING TEXT	39 WORDS

Security problems Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about

Security problems: Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

22/202	SUBMITTED TEXT	25 WORDS	77%	MATCHING TEXT	25 WORDS

various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an

various bank employees. They do not need access to information about customer accounts. In the file processing systems, as the application programs are added to the system in an

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

Difficulty in accessing data Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use. • Data isolation Because data is scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult. • Integrity problems The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is 8 compounded when constraints involve several data items from different files. • Atomicity problems A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system. • Concurrent-access anomalies For the sake of the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment,

SA 47F417BA15858476660.pdf (D123781188)

24/202	SUBMITTED TEXT	32 WORDS	100%	MATCHING TEXT	32 WORDS

customer, product, and purchase information. • Accounting: For payments, receipts, account balances, assets, and other accounting information. • Human resources: For information about employees, salaries, payroll taxes, and benefits, and for

customer, product, and purchase information. Accounting: For payments, receipts, account balances, assets and other accounting information. Human resources: For information about employees, salaries, payroll taxes, and benefits, and for

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

25/202	SUBMITTED TEXT	123 WORDS	100%	MATCHING TEXT	123 WORDS

production of items in factories, inventories of items in warehouses and stores, and orders for items. • Online retailers: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations. • Banking and Finance • Banking: For customer information, accounts, loans, and banking transactions. • Credit card transactions: For purchases on credit cards and generation of monthly statements. • Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm. Universities For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

production of items in factories, inventories of items in warehouses and stores, and orders for items. Online retailers: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations. Banking and Finance Banking: For customer information, accounts, loans, and banking transactions. Credit card transactions: For purchases on credit cards and generation of monthly statements. Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm. Universities: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

26/202	SUBMITTED TEXT	289 WORDS	99%	MATCHING TEXT	289 WORDS

interaction of concurrent updates may result in inconsistent data. Consider bank account A, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult 9 to provide because data may be accessed by many different application programs that have not been coordinated previously. • Security problems Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a bank enterprise as a running example of a typical data-processing application found in a corporation. 1.2.2 Components of

SA 47F417BA15858476660.pdf (D123781188)

27/202	SUBMITTED TEXT	14 WORDS	84%	MATCHING TEXT	14 WORDS

is the data model on which DBMS is based. The relational data model

SA 47F417BA15858476660.pdf (D123781188)

28/202	SUBMITTED TEXT	46 WORDS	79%	MATCHING TEXT	46 WORDS

data model. The relational DBMSs are evolving continuously, and have been incorporating many of the concepts that were developed in object 25 databases. This has led to a new class of DBMSs called object-relational DBMSs. Hence DBMSs can be categorized on the

SA 47F417BA15858476660.pdf (D123781188)

29/202	SUBMITTED TEXT	159 WORDS	97%	MATCHING TEXT	159 WORDS

data models: relational, object, object-relational, hierarchical, network, and other. The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with personal computers. Multiuser systems, which include the majority of DBMSs, support multiple users concurrently. A third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at multiple sites. A recent trend is to develop software to access several autonomous pre-existing databases stored under heterogeneous llBMSs. This leads to a federated DBMS (or

SA 47F417BA15858476660.pdf (D123781188)

30/202	SUBMITTED TEXT	22 WORDS	97%	MATCHING TEXT	22 WORDS

system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DBMSs use client- server architecture.

SA 47F417BA15858476660.pdf (D123781188)

31/202	SUBMITTED TEXT	48 WORDS	92%	MATCHING TEXT	48 WORDS

Entity-Relationship (ER): The entity-relationship (ER) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design. •

Entity-Relationship Model The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

32/202	SUBMITTED TEXT	30 WORDS	98%	MATCHING TEXT	30 WORDS

a collection of basic objects, called entities, and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects.

SA 47F417BA15858476660.pdf (D123781188)

33/202	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

can be seen as extending the E-R model with notions

SA 47F417BA15858476660.pdf (D123781188)

34/202	SUBMITTED TEXT	59 WORDS	100%	MATCHING TEXT	59 WORDS

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

35/202	SUBMITTED TEXT	16 WORDS	100%	MATCHING TEXT	16 WORDS

The object-relational data model combines features of the object-oriented data model and relational data model.

SA 47F417BA15858476660.pdf (D123781188)

36/202	SUBMITTED TEXT	16 WORDS	100%	MATCHING TEXT	16 WORDS

a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

SA 47F417BA15858476660.pdf (D123781188)

37/202	SUBMITTED TEXT	34 WORDS	95%	MATCHING TEXT	34 WORDS

is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL. The relational model is

is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL. The purpose the relational model is

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

38/202	SUBMITTED TEXT	22 WORDS	100%	MATCHING TEXT	22 WORDS

A relational database consists of a collection of tables, each of which is assigned 32 a unique name.

A relational database consists of a collection of tables, each of which is assigned a unique name.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

39/202	SUBMITTED TEXT	35 WORDS	100%	MATCHING TEXT	35 WORDS

relation In general, a row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of

relation. In general, a row in a table represents a relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

40/202	SUBMITTED TEXT	57 WORDS	100%	MATCHING TEXT	57 WORDS

table and the mathematical concept of relation, from which the relational data model takes its name. In mathematical terminology, a tuple is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an n-tuple of values, i.e., a tuple with n values, which corresponds to a row in a table.

table and the mathematical concept of relation, from which the relational data model takes its name. In mathematical terminology, a tuple is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an n-tuple of values, i.e., a tuple with n values, which corresponds to a row in a table.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

41/202	SUBMITTED TEXT	14 WORDS	88%	MATCHING TEXT	14 WORDS

there is a set of permitted values, called the domain of that attribute.

there is a set of permitted values, called the domain, or set, of that attribute.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

42/202	SUBMITTED TEXT	38 WORDS	100%	MATCHING TEXT	38 WORDS

We require that, for all relations r, the domains of all attributes of r be atomic. A domain is atomic if elements of the domain are considered to be indivisible units. For example, suppose the table instructor

We require that, for all relations r, the domains of all attributes of r be atomic. A domain is atomic if elements of the domain are considered to be indivisible units. For example, suppose the table instructor

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

43/202	SUBMITTED TEXT	54 WORDS	100%	MATCHING TEXT	54 WORDS

had an attribute phone_number, which can store a set of phone numbers corresponding to the 34 instructor. Then the domain of phone_number would not be atomic, since an element of the domain is a set of phone numbers, and it has subparts, namely the individual phone numbers in the set.

had an attribute phone number, which can store a set of phone numbers corresponding to the instructor. Then the domain of phone number would not be atomic, since an element of the domain is a set of phone numbers, and it has subparts, namely the individual phone numbers in the set.] –

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

44/202	SUBMITTED TEXT	21 WORDS	100%	MATCHING TEXT	21 WORDS

The null value is a special value that signifies that the value is unknown or does not exist. 2.4

The null value is a special value that signifies that the value is unknown or does not exist.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

45/202	SUBMITTED TEXT	20 WORDS	92%	MATCHING TEXT	20 WORDS

the relational model and uses a collection of tables to represent both data and the relationships among those data.

SA 47F417BA15858476660.pdf (D123781188)

46/202	SUBMITTED TEXT	31 WORDS	100%	MATCHING TEXT	31 WORDS

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. 42 •

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

47/202	SUBMITTED TEXT	16 WORDS	88%	MATCHING TEXT	16 WORDS

there is a set of permitted values, called the domain of that attribute. •

there is a set of permitted values, called the domain, or set, of that attribute.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

48/202	SUBMITTED TEXT	16 WORDS	96%	MATCHING TEXT	16 WORDS

A domain is atomic if elements of the domain are considered to be _____ units.

A domain is atomic if elements of the domain are considered to be indivisible units.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

49/202	SUBMITTED TEXT	178 WORDS	98%	MATCHING TEXT	178 WORDS

Occasionally database designers choose a schema that has redundant information; that is, it is not normalized. They use the redundancy to improve performance for specific applications. The penalty paid for not using a normalized schema is the extra work (in terms of coding time and execution time) to keep redundant data consistent. For instance, suppose all course prerequisites have to be displayed along with a course information, every time a course is accessed. In our normalized schema, this requires a join of course with prereq. One alternative to computing the join on the fly is to store a relation containing all the attributes of course and prereq. This makes displaying the "full" course information faster. However, the information for a course is repeated for every course prerequisite, and all copies must be updated by the application, whenever a course prerequisite is added or dropped. The process of taking a normalized schema and making it non normalized is called denormalization, and designers use it to tune performance of systems to support time-critical operations. 77

Occasionally database designers choose a schema that has redundant information; that is, it is not normalized. They use the redundancy to improve performance for specific applications. The penalty paid for not using a normalized schema is the extra work (in terms of coding time and execution time) to keep redundant data consistent. For instance, suppose all course prerequisites have to be displayed along with a course information, every time a course is accessed. In our normalized schema, this requires a join of course with prereq. One alternative to computing the join on the fly is to store a relation containing all the attributes of course and prereq. This makes displaying the "full" course information faster. However, the information for a course is repeated for every course prerequisite, and all copies must be updated by the application, whenever a course prerequisite is added or dropped. The process of taking a normalized schema and making it nonnormalized is called denormalization, and designers use it to tune performance of systems to support time-critical operations.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

50/202	SUBMITTED TEXT	18 WORDS	80%	MATCHING TEXT	18 WORDS

The process of taking a normalized schema and making it non normalized is called denormalization. 4.9

The process of taking a normalized schema and making it nonnormalized is called denormalization,

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

51/202	SUBMITTED TEXT	14 WORDS	80%	MATCHING TEXT	14 WORDS

of operations that form a single logical unit of work on a database.

of operations that form a single logical unit of work are transactions. A database

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

52/202	SUBMITTED TEXT	18 WORDS	100%	MATCHING TEXT	18 WORDS

Furthermore, it must manage concurrent execution of transactions in a way that avoids the introduction of inconsistency.

Furthermore, it must manage concurrent execution of transactions in a way that avoids the introduction of inconsistency. 4.1

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

53/202	SUBMITTED TEXT	47 WORDS	85%	MATCHING TEXT	47 WORDS

of Transaction Processing A transaction is defined as a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (like C++, or Java).

of inconsistency. 4.1 Transaction Concept A transaction is a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (for example, C++, or Java),

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

54/202	SUBMITTED TEXT	32 WORDS	88%	MATCHING TEXT	32 WORDS

The transaction consists of all operations executed between the begin transaction and end transaction. This collection of steps must appear to the user as a single, indivisible unit. If a transaction

The transaction consists of all operations executed between the begin transaction and end transaction. Atomicity: The collection of steps must appear to the user as a single, indivisible unit. Since a transaction

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

55/202	SUBMITTED TEXT	23 WORDS	100%	MATCHING TEXT	23 WORDS

begins to execute but fails for whatever reason, any changes to the database that the transaction may have made must be undone.

begins to execute but fails for whatever reason, any changes to the database that the transaction may have made must be undone.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

56/202	SUBMITTED TEXT	115 WORDS	83%	MATCHING TEXT	115 WORDS

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction. More precisely, there exists a set of waiting transactions {T 0 , T 1 , . . . , T n } such that T 0 is waiting for a data item that T 1 holds, and T 1 is waiting for a data item that T 2 holds, and T n-1 is waiting for a data item that T n holds, and T n is waiting for a data item that T 0 holds. None of the transactions can make progress in such a situation.

in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction More precisely, there exists a set of waiting transactions {T0, T1, . . . , Tn} such that T0 is waiting for a data item that T1 holds, and T1 is waiting for a data item that T2 holds, and . . . , and Tn–1 is waiting for a data item that Tn holds, and Tn is waiting for a data item that T0 holds. None of the transactions can make progress in such a situation.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

57/202	SUBMITTED TEXT	25 WORDS	92%	MATCHING TEXT	25 WORDS

the relational model and uses a collection of tables to represent both data and the relationships among those data. 42 •

SA 47F417BA15858476660.pdf (D123781188)

58/202	SUBMITTED TEXT	161 WORDS	88%	MATCHING TEXT	161 WORDS

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction. More precisely, there exists a set of waiting transactions {T 0 , T 1 , . . . , T n } such that T 0 is waiting for a data item that T 1 holds, and T 1 is waiting for a data item that T 2 holds, and T n-1 is waiting for a data item that T n holds, and T n is waiting for a data item that T 0 holds. None of the transactions can make progress in such a situation. The only solution to this undesirable situation is for the system to invoke some drastic action, such as rolling back some of the transactions involved in the deadlock. Rollback of a transaction may be partial: That is, a transaction may be rolled back to the

SA DBMS_AIML_FINAL.pdf (D111167082)

59/202	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

point where it obtained a lock whose release resolves the deadlock.

SA DBMS_AIML_FINAL.pdf (D111167082)

60/202	SUBMITTED TEXT	28 WORDS	94%	MATCHING TEXT	28 WORDS

prevention method is commonly used if the probability that the system would enter a deadlock state is relatively high; otherwise, detection and recovery are more efficient. 5.4.1

SA DBMS_AIML_FINAL.pdf (D111167082)

61/202	SUBMITTED TEXT	173 WORDS	95%	MATCHING TEXT	173 WORDS

Deadlock prevention offers two approaches to proceed. • First approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together. The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock, whenever the wait could potentially result in a deadlock. The simplest scheme under the first approach requires that each transaction locks all its data items before it begins execution. Moreover, either all are locked in one step or none are locked. There are two main disadvantages to this protocol: (1) it is often hard to predict, before the transaction begins, what data items need to be locked; (2) data-item utilization may be very low, since many of the data items may be locked but unused for a long time. • Second approach for preventing deadlocks is to impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering.

SA DBMS_AIML_FINAL.pdf (D111167082)

62/202	SUBMITTED TEXT	50 WORDS	59%	MATCHING TEXT	50 WORDS

preemption and transaction rollbacks. In preemption, when a transaction T j requests a lock that transaction T i holds, the lock granted to T i may be 96 preempted by rolling back of T i , and granting of the lock to

SA DBMS_AIML_FINAL.pdf (D111167082)

63/202	SUBMITTED TEXT	46 WORDS	89%	MATCHING TEXT	46 WORDS

If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed: (i) The wait–die scheme is a nonpreemptive technique. When transaction T i requests a data item currently held by

SA DBMS_AIML_FINAL.pdf (D111167082)

64/202	SUBMITTED TEXT	77 WORDS	76%	MATCHING TEXT	77 WORDS

If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed: (i) The wait–die scheme is a nonpreemptive technique. When transaction T i requests a data item currently held by T j , T i is allowed to wait only if it has a timestamp smaller than that of T j (that is, T i is older than

If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed: 1. The wait–die scheme is a non preemptive technique. When transaction Ti requests a data item currently held by Tj , is allowed to wait only if it has a timestamp smaller than that of Tj (that is, Ti is older than

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

65/202	SUBMITTED TEXT	26 WORDS	79%	MATCHING TEXT	26 WORDS

is allowed to wait only if it has a timestamp smaller than that of T j (that is, T i is older than

SA DBMS_AIML_FINAL.pdf (D111167082)

66/202	SUBMITTED TEXT	17 WORDS	88%	MATCHING TEXT	17 WORDS

The wound–wait scheme is a preemptive technique. It is opposite to the wait–die scheme. When transaction

SA DBMS_AIML_FINAL.pdf (D111167082)

67/202	SUBMITTED TEXT	80 WORDS	66%	MATCHING TEXT	80 WORDS

is rolled back (dies). (ii) The wound–wait scheme is a preemptive technique. It is opposite to the wait–die scheme. When transaction T i requests a data item currently held by T j , T i is allowed to wait only if it has a timestamp larger than that of T j (that is, T i is younger than T j). Otherwise, T j is rolled back (T j is wounded by

is rolled back (dies). 2. The wound–wait scheme is a preemptive technique. It is a counterpart to the wait–die scheme. When transaction Ti requests a data item currently held by Tj , is allowed to wait only if it has a timestamp larger than that of Tj (that is, Ti is younger than Tj). Otherwise, Tj is rolled back (Tj is wounded by

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

68/202	SUBMITTED TEXT	20 WORDS	83%	MATCHING TEXT	20 WORDS

A transaction is defined as a unit of program execution that accesses and possibly updates various data items. •

A transaction is a unit of program execution that accesses and possibly updates various data items.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

69/202	SUBMITTED TEXT	114 WORDS	100%	MATCHING TEXT	114 WORDS

Atomicity ensures that either all the effects of a transaction are reflected in the database, or none are; a failure cannot leave the database in a state where a transaction is partially executed. • Consistency ensures that, if the database is initially consistent, the execution of the transaction (by itself) leaves the database in a consistent state. 103 • Isolation ensures that concurrently executing transactions are isolated from one another, so that each has the impression that no other transaction is executing concurrently with it. • Durability ensures that, once a transaction has been committed, that transaction's updates do not get lost, even if there is a system failure. •

Atomicity ensures that either all the effects of a transaction are reflected in the database, or none are; a failure cannot leave the database in a state where a transaction is partially executed. 2. Consistency ensures that, if the database is initially consistent, the execution of the transaction (by itself) leaves the database in a consistent state. 3. Isolation ensures that concurrently executing transactions are isolated from one another, so that each has the impression that no other transaction is executing concurrently with it. 4. Durability ensures that, once a transaction has been committed, that transaction's updates do not get lost, even if there is a system failure. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

70/202	SUBMITTED TEXT	26 WORDS	65%	MATCHING TEXT	26 WORDS

is allowed to wait only if it has a timestamp larger than that of T j (that is, T i is younger than

SA DBMS_AIML_FINAL.pdf (D111167082)

71/202	SUBMITTED TEXT	21 WORDS	92%	MATCHING TEXT	21 WORDS

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction.

in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

72/202	SUBMITTED TEXT	21 WORDS	92%	MATCHING TEXT	21 WORDS

in a deadlock state, if there exists a set of transactions such that every transaction is waiting for another transaction.

SA DBMS_AIML_FINAL.pdf (D111167082)

73/202	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

Block movements between disk and main memory are initiated through

Block movements between disk and main memory are initiated through

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

74/202	SUBMITTED TEXT	31 WORDS	100%	MATCHING TEXT	31 WORDS

two operations: 1. input(B) transfers the physical block B to main memory. 2. output(B) transfers the buffer block B to the disk, and replaces the appropriate physical block there.

two operations: 1. input(B) transfers the physical block B to main memory. 2. output(B) transfers the buffer block B to the disk, and replaces the appropriate physical block there.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

75/202	SUBMITTED TEXT	71 WORDS	95%	MATCHING TEXT	71 WORDS

Therefore, the DBMS must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is a recovery system that can restore the database to the consistent state that existed before the failure. The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a failure.

SA DBMS_AIML_FINAL.pdf (D111167082)

76/202	SUBMITTED TEXT	27 WORDS	100%	MATCHING TEXT	27 WORDS

Failure Classification There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner.

SA DBMS_AIML_FINAL.pdf (D111167082)

77/202	SUBMITTED TEXT	84 WORDS	93%	MATCHING TEXT	84 WORDS

Transaction failure: There are two types of errors that may cause a transaction to fail: • Logical error • : The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded. System error 2. System crash: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt.

Transaction failure. There are two types of errors that may cause a transaction to fail: The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded. • System error. The system has entered an undesirable state (for example, deadlock). crash. There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

78/202	SUBMITTED TEXT	50 WORDS	98%	MATCHING TEXT	50 WORDS

Transaction failure: There are two types of errors that may cause a transaction to fail: • Logical error • : The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded. System error 2. System

SA DBMS_AIML_FINAL.pdf (D111167082)

79/202	SUBMITTED TEXT	112 WORDS	100%	MATCHING TEXT	112 WORDS

System crash: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of nonvolatile storage remains intact, and is not corrupted. The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption. Well- designed systems have numerous internal checks, at the hardware and the software level, that bring the system to a halt when there is an error. Hence, the fail-stop assumption is a reasonable one. :

SA DBMS_AIML_FINAL.pdf (D111167082)

80/202	SUBMITTED TEXT	30 WORDS	100%	MATCHING TEXT	30 WORDS

deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be re-executed at a later time. 116 3.

SA DBMS_AIML_FINAL.pdf (D111167082)

81/202	SUBMITTED TEXT	31 WORDS	100%	MATCHING TEXT	31 WORDS

Disk failure: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or

Disk failure. A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

82/202	SUBMITTED TEXT	14 WORDS	100%	MATCHING TEXT	14 WORDS

media, such as DVD or tapes, are used to recover from the failure.

media, such as DVD or tapes, are used to recover from the failure. 4.15.1

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

83/202	SUBMITTED TEXT	31 WORDS	100%	MATCHING TEXT	31 WORDS

Disk failure: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or

SA DBMS_AIML_FINAL.pdf (D111167082)

84/202	SUBMITTED TEXT	24 WORDS	65%	MATCHING TEXT	24 WORDS

media, such as DVD or tapes, are used to recover from the failure. There is a need to identify the failure modes of

SA DBMS_AIML_FINAL.pdf (D111167082)

85/202	SUBMITTED TEXT	28 WORDS	75%	MATCHING TEXT	28 WORDS

how these failure modes affect the contents of the database. Accordingly, algorithms can be proposed to ensure database consistency and transaction atomicity despite failures. These algorithms

SA DBMS_AIML_FINAL.pdf (D111167082)

86/202	SUBMITTED TEXT	47 WORDS	100%	MATCHING TEXT	47 WORDS

have two parts: 1. Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures. 2. Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability. 6.6 Recovery

SA DBMS_AIML_FINAL.pdf (D111167082)

Shadow paging considers the database to be made up of n number of fixed-size disk pages (or disk blocks) for recovery purposes. A directory with n entries is constructed, where the i th entry points to the i th database page on disk. The directory is kept in main memory if it is not too large, and all references, reads or writes, to database pages on disk go through it. When a transaction begins executing, the current directory, whose entries point to the most recent or current database pages on disk, is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory. To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database 121 thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery. In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits.

SA DBMS_AIML_FINAL.pdf (D111167082)

Shadow paging considers the database to be made up of n number of fixed-size disk pages (or disk blocks) for recovery purposes. A directory with n entries is constructed, where the i th entry points to the i th database page on disk. The directory is kept in main memory if it is not too large, and all references, reads or writes, to database pages on disk go through it. When a transaction begins executing, the current directory, whose entries point to the most recent or current database pages on disk, is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere—on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory. To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database 121 thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery. In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits. The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation. 6.7

SA 47F417BA15858476660.pdf (D123781188)

89/202	SUBMITTED TEXT	16 WORDS	76%	MATCHING TEXT	16 WORDS

remote backup system When the primary site fails, the remote backup site takes over processing.

remote backup site. If the primary site fails, the remote backup site takes over transaction processing,

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

90/202	SUBMITTED TEXT	31 WORDS	100%	MATCHING TEXT	31 WORDS

Several issues must be addressed in designing a remote backup system: • Detection of failure: It is important for the remote backup system to detect when the primary has failed.

Several issues must be addressed in designing a remote backup system: • Detection of failure. It is important for the remote backup system to detect when the primary has failed.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

91/202	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

with independent modes of failure between the primary and the remote backup.

with independent modes of failure between the primary and the remote backup. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

92/202	SUBMITTED TEXT	43 WORDS	100%	MATCHING TEXT	43 WORDS

Transfer of control: When the primary fails, the backup site takes over processing and becomes the new primary. When the original primary site recovers, it can either play the role of remote backup, or take over the role of primary site again.

Transfer of control. When the primary fails, the backup site takes over processing and becomes the new primary. When the original primary site recovers, it can either play the role of remote backup, or take over the role of primary site again. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

93/202	SUBMITTED TEXT	70 WORDS	100%	MATCHING TEXT	70 WORDS

Time to recover: If the log at the remote backup grows large, recovery will take a long time. The remote backup site can periodically process the redo log records that it has received and can perform a checkpoint, so that earlier parts of the log can be deleted. The delay before the remote backup takes over can be significantly reduced as a result. • A hot-spare configuration can make

Time to recover. If the log at the remote backup grows large, recovery will take a long time. The remote backup site can periodically process the redo log records that it has received and can perform a checkpoint, so that earlier parts of the log can be deleted. The delay before the remote backup takes over can be significantly reduced as a result. A hot-spare configuration can make

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

94/202	SUBMITTED TEXT	83 WORDS	100%	MATCHING TEXT	83 WORDS

Time to commit: To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site. This delay can result in a longer wait to commit a transaction, and some systems therefore permit lower degrees of durability. The degrees of durability can be classified as follows: ? One-safe ? : A transaction commits as soon as its commit log record is written to stable storage at the primary site.

Time to commit. To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site. This delay can result in a longer wait to commit a transaction, and some systems therefore permit lower degrees of durability. The degrees of durability can be classified as follows: • One-safe. A transaction commits as soon as its commit log record is written to stable storage at the primary site. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

95/202	SUBMITTED TEXT	30 WORDS	85%	MATCHING TEXT	30 WORDS

Two-very-safe: A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site. The problem with this scheme is

Two-very-safe. A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site. • Two-safe. This scheme is

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

96/202	SUBMITTED TEXT	64 WORDS	100%	MATCHING TEXT	64 WORDS

This scheme is the same as two-very-safe if both primary and backup sites are active. If only the primary is active, the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site. This scheme provides better availability than does two-very-safe, while avoiding the problem of lost transactions faced by the one-safe scheme.

This scheme is the same as two-very-safe if both primary and backup sites are active. If only the primary is active, the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site. This scheme provides better availability than does two-very-safe, while avoiding the problem of lost transactions faced by the one-safe scheme.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

97/202	SUBMITTED TEXT	40 WORDS	100%	MATCHING TEXT	40 WORDS

An alternative way of achieving high availability is to use a distributed database, with data replicated at more than one site. Transactions are then required to update all replicas of any data item that they update. 6.9

An alternative way of achieving high availability is to use a distributed database, with data replicated at more than one site. Transactions are then required to update all replicas of any data item that they update.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

98/202	SUBMITTED TEXT	13 WORDS	87%	MATCHING TEXT	13 WORDS

An integral part of a database system is a recovery system that

An integral part of a database system is a recovery that

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

99/202	SUBMITTED TEXT	59 WORDS	100%	MATCHING TEXT	59 WORDS

The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation. 6.7

SA DBMS_AIML_FINAL.pdf (D111167082)

100/202	SUBMITTED TEXT	22 WORDS	100%	MATCHING TEXT	22 WORDS

In addition to system failures, transactions may also fail for various reasons, such as violation of integrity constraints or deadlocks. •

In addition to system failures, transactions may also fail for various reasons, such as violation of integrity constraints or deadlocks. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

101/202	SUBMITTED TEXT	70 WORDS	97%	MATCHING TEXT	70 WORDS

The various types of storage in a computer are volatile storage, nonvolatile storage, and stable storage. Data in volatile storage, such as in RAM , are lost 127 when the computer crashes. Data in nonvolatile storage, such as disk, are not lost when the computer crashes, but may occasionally be lost because of failures such as disk crashes. Data in stable storage is never lost. •

The various types of storage in a computer are volatile storage, nonvolatile storage, and stable storage. Data in volatile storage, such as in RAM, are lost when the computer crashes. Data in nonvolatile storage, such as disk, are not lost when the computer crashes, but may occasionally be lost because of failures such as disk crashes. Data in stable storage are never lost. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

102/202	SUBMITTED TEXT	28 WORDS	94%	MATCHING TEXT	28 WORDS

An integral part of a database system is a recovery system that can restore the database to the consistent state that existed before the failure. •

SA DBMS_AIML_FINAL.pdf (D111167082)

103/202	SUBMITTED TEXT	13 WORDS	95%	MATCHING TEXT	13 WORDS

a state of the world that the database is supposed to capture.

SA 47F417BA15858476660.pdf (D123781188)

104/202	SUBMITTED TEXT	60 WORDS	100%	MATCHING TEXT	60 WORDS

In case of failure, the state of the database system may no longer be consistent; that is, it may not reflect a state of the world that the database is supposed to capture. To preserve consistency, we require that each transaction be atomic. It is the responsibility of the recovery scheme to ensure the atomicity and durability property. •

In case of failure, the state of the database system may no longer be consistent; that is, it may not reflect a state of the world that the database is supposed to capture. To preserve consistency, we require that each transaction be atomic. It is the responsibility of the recovery scheme to ensure the atomicity and durability property. 4.18.1

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

105/202	SUBMITTED TEXT	47 WORDS	100%	MATCHING TEXT	47 WORDS

In log-based schemes, all updates are recorded on a log, which must be kept in stable storage. A transaction is considered to have committed when its last log record, which is the commit log record for the transaction, has been output to stable storage. •

In log-based schemes, all updates are recorded on a log, which must be kept in stable storage. A transaction is considered to have committed when its last log record, which is the commit log record for the transaction, has been output to stable storage.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

106/202	SUBMITTED TEXT	41 WORDS	92%	MATCHING TEXT	41 WORDS

Remote backup systems provide a high degree of availability, allowing transaction processing to continue even if the primary site is destroyed. Data and log records from a primary site are continually backed up to a remote backup site. 6.10

Remote backup systems provide a high degree of availability, allowing transaction processing to continue even if the primary site is destroyed by a fire, flood, or earthquake. Data and log records from a primary site are continually backed up to a remote backup site.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

107/202	SUBMITTED TEXT	13 WORDS	95%	MATCHING TEXT	13 WORDS

a state of the world that the database is supposed to capture.

SA DBMS_AIML_FINAL.pdf (D111167082)

108/202	SUBMITTED TEXT	17 WORDS	68%	MATCHING TEXT	17 WORDS

When many transactions execute concurrently in the database, the isolation property may no longer be preserved.

When several transactions execute concurrently in the database, the consistency of data may no longer be preserved.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

109/202	SUBMITTED TEXT	221 WORDS	90%	MATCHING TEXT	221 WORDS

W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. • R-timestamp(Q) denotes the largest timestamp of any transaction that executed read(Q) successfully. These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed. The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows: 1. Suppose that transaction T i issues read(Q). a) If TS(T i) > W-timestamp(Q), then T i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T i is rolled back. b) If TS(T i) ≥ W-timestamp(Q), then the read operation is executed, and R- timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T i). 2. Suppose that transaction T i issues write(Q). a) If TS(T i) > R-timestamp(Q), then the value of Q that T i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls T i back. b) If TS(T i) > W-timestamp(Q), then T i is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls T i back. c) Otherwise, the system executes

W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. timestamp(Q) denotes the largest timestamp of any transaction that executed read(Q) successfully.These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.4.13.1 The Timestamp-Ordering Protocol: The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows: that transaction Ti issues read(Q). a. If TS(Ti) > W-timestamp(Q), then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back. b. If TS(Ti) ≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(Ti). 2. Suppose that transaction Ti issues write(Q). If TS(Ti) > R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls Ti back. b. If TS(Ti) > W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls Ti back. c. Otherwise, the system executes

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

110/202	SUBMITTED TEXT	48 WORDS	93%	MATCHING TEXT	48 WORDS

write operation and sets W- timestamp(Q) to TS(T i). 137 If a transaction T i is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

write operation and sets W-timestamp(Q) to TS(Ti). If a transaction Ti is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

111/202	SUBMITTED TEXT	17 WORDS	100%	MATCHING TEXT	17 WORDS

The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order.

The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order. 4.13.2

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

112/202	SUBMITTED TEXT	32 WORDS	100%	MATCHING TEXT	32 WORDS

The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the fail-stop assumption. •

SA DBMS_AIML_FINAL.pdf (D111167082)

113/202	SUBMITTED TEXT	92 WORDS	88%	MATCHING TEXT	92 WORDS

Suppose that transaction T i issues write(Q). 1. If TS(T i) > R-timestamp(Q), then the value of Q that T i is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls T i back. 2. If TS(T i) > W-timestamp(Q), then T i is attempting to write an obsolete value of Q. Hence, this write operation can be ignored. 3. Otherwise, the system executes the write operation and

Suppose that transaction Ti issues write(Q). 1. If TS(Ti) > R-timestamp(Q), then the value of Q that Ti is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls Ti back. 2. If TS(Ti) > W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation can be ignored. 3. Otherwise, the system executes the write operation and

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

114/202	SUBMITTED TEXT	17 WORDS	91%	MATCHING TEXT	17 WORDS

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions

in cases where a majority of transactions are read-only transactions, and thus the rate of conflicts among these transactions

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

115/202	SUBMITTED TEXT	25 WORDS	77%	MATCHING TEXT	25 WORDS

However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction.

SA DBMS_AIML_FINAL.pdf (D111167082)

116/202	SUBMITTED TEXT	169 WORDS	92%	MATCHING TEXT	169 WORDS

The validation protocol requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order: 1. Read phase: During this phase, the system executes transaction T i . It reads the values of the various data items and stores them in variables local to T i . It performs all write operations on temporary local variables, without updates of the actual database. 2. Validation phase: The validation test is applied to transaction T i . This determines whether T i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction. 3. Write phase: If the validation test succeeds for transaction T i , the temporary local variables that hold the results of any write operations performed by T i are copied to the database. Read-only transactions omit this phase.

The validation protocol requires that each transaction Ti executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order: 1. Read phase. During this phase, the system executes transaction Ti. It reads the values of the various data items and stores them in variables local to Ti. It performs all write operations on temporary local variables, without updates of the actual database. 2. Validation phase. The validation test (described below) is applied to transaction Ti . This determines whether Ti is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction. 3. Write phase. If the validation test succeeds for transaction Ti, the temporary local variables that hold the results of any write operations performed by Ti are copied to the database. Read-only transactions omit this phase.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

117/202	SUBMITTED TEXT	271 WORDS	95%	MATCHING TEXT	271 WORDS

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low. Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state. A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead. A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for monitoring the system. The validation protocol requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order: 1. Read phase: During this phase, the system executes transaction T i . It reads the values of the various data items and stores them in variables local to T i . It performs all write operations on temporary local variables, without updates of the actual database. 2. Validation phase: The validation test is applied to transaction T i . This determines whether T i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction. 3. Write phase: If the validation test succeeds for transaction T i , the temporary local variables that hold the results of any write operations performed by T i are copied to the database. Read-only transactions omit this phase.

SA DBMS_AIML_FINAL.pdf (D111167082)

118/202	SUBMITTED TEXT	69 WORDS	89%	MATCHING TEXT	69 WORDS

Each transaction must go through the phases in the order shown. However, phases of concurrently executing transactions can be interleaved. To perform the validation test, we need to know when the various phases of transactions took 139 place. We shall, therefore, associate three different timestamps with each transaction T i : • Start(T i), the time when T i started its execution. • Validation(

Each transaction must go through the phases in the order shown. However, phases of concurrently executing transactions can be interleaved. To perform the validation test, we need to know when the various phases of transactions took place. We shall, theefore, associate three different timestamps with each transaction Ti : 1. Start(Ti), the time when Ti started its execution. 2. Validation(

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

119/202	SUBMITTED TEXT	69 WORDS	92%	MATCHING TEXT	69 WORDS

Each transaction must go through the phases in the order shown. However, phases of concurrently executing transactions can be interleaved. To perform the validation test, we need to know when the various phases of transactions took 139 place. We shall, therefore, associate three different timestamps with each transaction T i : • Start(T i), the time when T i started its execution. • Validation(

SA DBMS_AIML_FINAL.pdf (D111167082)

120/202	SUBMITTED TEXT	17 WORDS	89%	MATCHING TEXT	17 WORDS

the time when T i finished its read phase and started its validation phase. • Finish(

the time when Ti finished its read phase and started its validation phase. 3. Finish(

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

121/202	SUBMITTED TEXT	17 WORDS	89%	MATCHING TEXT	17 WORDS

the time when T i finished its read phase and started its validation phase. • Finish(

SA DBMS_AIML_FINAL.pdf (D111167082)

122/202	SUBMITTED TEXT	15 WORDS	76%	MATCHING TEXT	15 WORDS

equivalent to a serial schedule in which transaction T j appears before transaction

equivalent to a serial schedule in which transaction Ti appears before transaction

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

123/202	SUBMITTED TEXT	41 WORDS	75%	MATCHING TEXT	41 WORDS

the time when T i finished its write phase. We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation(T i). Thus, the value TS(T i) = Validation(T i) and, if TS(

SA DBMS_AIML_FINAL.pdf (D111167082)

124/202	SUBMITTED TEXT	34 WORDS	61%	MATCHING TEXT	34 WORDS

then any produced schedule must be equivalent to a serial schedule in which transaction T j appears before transaction T k . The reason we have chosen Validation(T i), rather than Start(

SA DBMS_AIML_FINAL.pdf (D111167082)

125/202	SUBMITTED TEXT	34 WORDS	62%	MATCHING TEXT	34 WORDS

The validation test for transaction T i requires that, for all transactions T k with TS(T k) > TS(T i), one of the following two conditions must hold: 1. Finish(

The validation test for transaction Ti requires that, for all transactions Tk with TS(Tk) > TS(Ti), one of the following two conditions must hold: 1. Finish(

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

126/202	SUBMITTED TEXT	39 WORDS	91%	MATCHING TEXT	39 WORDS

as the timestamp of transaction T i is that we can expect faster response time provided that conflict rates among transactions are indeed low. The validation test for transaction T i requires that, for all transactions

SA DBMS_AIML_FINAL.pdf (D111167082)

127/202	SUBMITTED TEXT	20 WORDS	64%	MATCHING TEXT	20 WORDS

the serializability order is indeed maintained. 2. The set of data items written by T k does not

the serializability order is indeed maintained. 2. The set of data itemswritten by Tk does not

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

128/202	SUBMITTED TEXT	20 WORDS	64%	MATCHING TEXT	20 WORDS

the serializability order is indeed maintained. 2. The set of data items written by T k does not

SA DBMS_AIML_FINAL.pdf (D111167082)

129/202	SUBMITTED TEXT	59 WORDS	43%	MATCHING TEXT	59 WORDS

Validation(T i)). This condition ensures that the writes of T k and T i do not overlap. Since the writes of T k do not affect the read of T i , and since T i cannot affect the read of T k , the serializability order is indeed maintained. The validation scheme

Validation(Ti)). This condition ensures that the writes of Tk and Ti do not overlap. Since the writes of Tk do not affect the read of Ti , and since Ti cannot affect the read of Tk , the serializability order is indeed maintained.This validation scheme

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

130/202	SUBMITTED TEXT	27 WORDS	100%	MATCHING TEXT	27 WORDS

This validation scheme is called the optimistic concurrency-control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end.

This validation scheme is called the optimistic concurrency-control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end. 4.15

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

131/202	SUBMITTED TEXT	56 WORDS	43%	MATCHING TEXT	56 WORDS

Validation(T i)). This condition ensures that the writes of T k and T i do not overlap. Since the writes of T k do not affect the read of T i , and since T i cannot affect the read of T k , the serializability order is indeed maintained.

SA DBMS_AIML_FINAL.pdf (D111167082)

132/202	SUBMITTED TEXT	23 WORDS	87%	MATCHING TEXT	23 WORDS

to be read. The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability.

to be read. The concurrency-control scheme ensures that the version to be read is selected in a manner that ensures serializability,

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

133/202	SUBMITTED TEXT	18 WORDS	81%	MATCHING TEXT	18 WORDS

With each transaction T i in the system, we associate a unique static timestamp, denoted by TS(

With each transaction Ti in the system, we associate a unique fixed timestamp, denoted by TS(

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

134/202	SUBMITTED TEXT	16 WORDS	78%	MATCHING TEXT	16 WORDS

contains three data fields: • Content is the value of version Q k . • W-timestamp(

contains three data fields: • Content is the value of version Qk . • W-timestamp(

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

135/202	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

is the largest timestamp of any transaction that successfully read version

is the largest timestamp of any transaction that successfully read version

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

136/202	SUBMITTED TEXT	41 WORDS	100%	MATCHING TEXT	41 WORDS

To ensure serializability, we can use various concurrency-control schemes. All these schemes either delay an operation or abort the transaction that issued the operation. The most common ones are locking protocols, timestamp ordering schemes, validation techniques, and multiversion schemes. •

To ensure serializability, we can use various concurrency-control schemes. All these schemes either delay an operation or abort the transaction that issued the operation. The most common ones are locking protocols, timestamp ordering schemes, validation techniques, and multiversion schemes. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

137/202	SUBMITTED TEXT	27 WORDS	76%	MATCHING TEXT	27 WORDS

A timestamp-ordering scheme ensures serializability by selecting an ordering in advance between every pair of transactions. The timestamps of the transactions determine the serializability order. •

A timestamp-ordering scheme ensures serializability by selecting an ordering in advance between every pair of transactions. A unique fixed timestamp is associated with each transaction in the system. The timestamps of the transactions determine the serializability order.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

138/202	SUBMITTED TEXT	133 WORDS	97%	MATCHING TEXT	133 WORDS

The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed. However, there is a possibility of starvation of long transactions, due to a sequence of conflicting short transactions that cause repeated restarts of the 140 long transaction. To avoid starvation, conflicting transactions must be temporarily blocked, to enable the long transaction to finish. This validation scheme is called the optimistic concurrency-control scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end. In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conflict is detected, even though there is a chance that the schedule may be conflict serializable. 7.5

SA DBMS_AIML_FINAL.pdf (D111167082)

139/202	SUBMITTED TEXT	30 WORDS	100%	MATCHING TEXT	30 WORDS

A validation scheme is an appropriate concurrency-control method in cases where a majority of transactions are read-only transactions, and thus the rate of conflicts among these transactions is low.

A validation scheme is an appropriate concurrency-control method in cases where a majority of transactions are read-only transactions, and thus the rate of conflicts among these transactions is low.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

140/202	SUBMITTED TEXT	21 WORDS	100%	MATCHING TEXT	21 WORDS

The serializability order is determined by the timestamp of the transaction. A transaction in this scheme is never delayed. •

The serializability order is determined by the timestamp of the transaction. A transaction in this scheme is never delayed.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

141/202	SUBMITTED TEXT	42 WORDS	100%	MATCHING TEXT	42 WORDS

A multiversion concurrency-control scheme is based on the creation of a new version of a data item for each transaction that writes that item. When a read operation is issued, the system selects one of the versions to be read. •

A multiversion concurrency-control scheme is based on the creation of a new version of a data item for each transaction that writes that item. When a read operation is issued, the system selects one of the versions to be read.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

142/202	SUBMITTED TEXT	43 WORDS	100%	MATCHING TEXT	43 WORDS

Snapshot isolation is a multiversion concurrency-control protocol based on validation, which, unlike multiversion two-phase locking, does not require transactions to be declared as read-only or update. Snapshot isolation does not guarantee serializability, but is nevertheless supported by many database systems. 7.8

Snapshot isolation is a multiversion concurrency-control protocol based on validation, which, unlike multiversion two-phase locking, does not require transactions to be declared as read-only or update. Snapshot isolation does not guarantee serializability, but is nevertheless supported by many database systems. 4.17

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

143/202	SUBMITTED TEXT	29 WORDS	90%	MATCHING TEXT	29 WORDS

W-timestamp(Q): It denotes the largest timestamp of any transaction that executed write(Q) successfully. • R-timestamp(Q): It denotes the largest timestamp of any transaction that executed read(Q) successfully. •

W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. Q) denotes the largest timestamp of any transaction that executed read(Q) successfully.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

144/202	SUBMITTED TEXT	11 WORDS	100%	MATCHING TEXT	11 WORDS

in cases where a majority of transactions are read-only transactions,

SA DBMS_AIML_FINAL.pdf (D111167082)

145/202	SUBMITTED TEXT	32 WORDS	94%	MATCHING TEXT	32 WORDS

requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. 7.9

requires that each transaction Ti executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

146/202	SUBMITTED TEXT	18 WORDS	64%	MATCHING TEXT	18 WORDS

When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.

When several transactions execute concurrently in the database, the consistency of data may no longer be preserved.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

147/202	SUBMITTED TEXT	70 WORDS	100%	MATCHING TEXT	70 WORDS

One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. 8.1

One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. 4.10.1

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

148/202	SUBMITTED TEXT	24 WORDS	91%	MATCHING TEXT	24 WORDS

Shared: If a transaction T i has obtained a shared-mode lock (denoted by S) 148 on item Q, then

Shared. If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

149/202	SUBMITTED TEXT	27 WORDS	93%	MATCHING TEXT	27 WORDS

can read, but cannot write, Q. 2. Exclusive: If a transaction T i has obtained an exclusive-mode lock (denoted by X) on item Q, then

can read, but cannot write, Q. 2. Exclusive. If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

150/202	SUBMITTED TEXT	90 WORDS	98%	MATCHING TEXT	90 WORDS

Two-Phase Locking (2PL) One protocol that ensures serializability is the two-phase locking protocol. This protocol requires that each transaction issue lock and unlock requests in two phases: 1. Growing phase: A transaction may obtain locks, but may not release any lock. 2. Shrinking phase: A transaction may release locks, but may not obtain any new locks. Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

Two-Phase Locking Protocol One protocol that ensures serializability is the two-phase locking protocol. This protocol requires that each transaction issue lock and unlock requests in two phases: 1. Growing phase. A transaction may obtain locks, but may not release any lock. 2. Shrinking phase. A transaction may release locks, but may not obtain any new locks. Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

151/202	SUBMITTED TEXT	28 WORDS	100%	MATCHING TEXT	28 WORDS

The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the lock point of the transaction.

The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the lock point of the transaction.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

152/202	SUBMITTED TEXT	18 WORDS	100%	MATCHING TEXT	18 WORDS

Cascading rollbacks can be avoided by a modification of two-phase locking called the strict two-phase locking protocol.

Cascading rollbacks can be avoided by a modification of two-phase locking called the strict two-phase locking protocol.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

153/202	SUBMITTED TEXT	23 WORDS	100%	MATCHING TEXT	23 WORDS

Another variant of two-phase locking is the rigorous two-phase locking protocol, which requires that all locks be held until the transaction commits.

Another variant of two-phase locking is the rigorous two-phase locking protocol, which requires that all locks be held until the transaction commits.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

154/202	SUBMITTED TEXT	45 WORDS	100%	MATCHING TEXT	45 WORDS

Implementation of Locking A lock manager can be implemented as a process that receives messages from transactions and sends messages in reply. The lock-manager process replies to lock-request messages with lock-grant messages, or with messages requesting 154 rollback of the transaction (

Implementation of Locking: A lock manager can be implemented as a process that receives messages from transactions and sends messages in reply. The lock-manager process replies to lock-request messages with lock-grant messages, or with messages requesting rollback of the transaction .

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

155/202	SUBMITTED TEXT	118 WORDS	100%	MATCHING TEXT	118 WORDS

Unlock messages require only an acknowledgment in response, but may result in a grant message to another waiting transaction. The lock manager uses this data structure: For each data item that is currently locked, it maintains a linked list of records, one for each request, in the order in which the requests arrived. It uses a hash table, indexed on the name of a data item, to find the linked list (if any) for a data item; this table is called the lock table. Each record of the linked list for a data item notes which transaction made the request, and what lock mode it requested. The record also notes if the request has currently been granted.

Unlock messages require only an acknowledgment in response, but may result in a grant message to another waiting transaction. The lock manager uses this data structure: For each data item that is currently locked, it maintains a linked list of records, one for each request, in the order in which the requests arrived. It uses a hash table, indexed on the name of a data item, to find the linked list (if any) for a data item; this table is called the lock table. Each record of the linked list for a data item notes which transaction made the request, and what lock mode it requested. The record also notes if the request has currently been granted.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

156/202	SUBMITTED TEXT	217 WORDS	98%	MATCHING TEXT	217 WORDS

The lock manager processes requests this way: • When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present. Otherwise it creates a new linked list, containing only the record for the request. It always grants a lock request on a data item that is not currently locked. But if the transaction requests a lock on an item on which a lock is currently held, the lock manager grants the request only if it is compatible with the locks that are currently held, and all earlier requests have been granted already. Otherwise the request has to wait. • When the lock manager receives an unlock message from a transaction, it deletes the record for that data item in the linked list corresponding to that transaction. It tests the record that follows, if any, to see if that request can now be granted. If it can, the lock manager grants that request, and processes the record following it, if any, similarly, and so on. • If a transaction aborts, the lock manager deletes any waiting request made by the transaction. Once the database system has taken appropriate actions to undo the transaction, it releases all locks held by the aborted transaction.

The lock manager processes requests this way: • When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present. Otherwise it creates a new linked list, containing only the record for the request. It always grants a lock request on a data item that is not currently locked. But if the transaction requests a lock on an item on which a lock is currently held, the lock manager grants the request only if it is compatible with the locks that are currently held, and all earlier requests have been granted already. Otherwise the request has to wait. • When the lock manager receives an unlock message from a transaction, it deletes the record for that data item in the linked list corresponding to that transaction. It tests the record that follows, if any, as described in the previous paragraph, to see if that request can now be granted. If it can, the lock manager grants that request, and processes the record following it, if any, similarly, and so on. • If a transaction aborts, the lock manager deletes any waiting request made by the transaction. Once the database system has taken appropriate actions to undo the transaction, it releases all locks held by the aborted transaction. 4.10.5

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

157/202 **SUBMITTED TEXT** 183 WORDS **88%** **MATCHING TEXT** 183 WORDS

The multiple-granularity locking protocol uses these lock modes to ensure serializability. It requires that a transaction T i that attempts to lock a node Q must follow these rules: 1. Transaction T i must observe the lock-compatibility function of Figure 8.4. 2. Transaction T i must lock the root of the tree first, and can lock it in any mode. 3. Transaction T i can lock a node Q in S or IS mode only if T i currently has the parent of Q locked in either IX or IS mode. 4. Transaction T i can lock a node Q in X, SIX , or IX mode only if T i currently has the parent of Q locked in either IX or SIX mode. 5. Transaction T i can lock a node only if T i has not previously unlocked any node (that is, T i is two phase). 6. Transaction T i can unlock a node Q only if T i currently has none of the children of Q locked.

The multiple-granularity locking protocol uses these lock modes to ensure serializability. It requires that a transaction Ti that attempts to lock a node Q must follow these rules: 1. Transaction Ti must observe the lock-compatibility function of Figure 4.12. 2. Transaction Ti must lock the root of the tree first, and can lock it in any mode. 3. Transaction Ti can lock a node Q in S or IS mode only if Ti currently has the parent of Q locked in either IX or IS mode. 4. Transaction Ti can lock a node Q in X, SIX, or IX mode only if Ti currently has the parent of Q locked in either IX or SIX mode. 5. Transaction Ti can lock a node only if Ti has not previously unlocked any node (that is, Ti is two phase). 6. Transaction Ti can unlock a node Q only if Ti currently has none of the children of Q locked.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

158/202 **SUBMITTED TEXT** 25 WORDS **100%** **MATCHING TEXT** 25 WORDS

that the multiple-granularity protocol requires that locks be acquired in top-down (root-to-leaf) order, whereas locks must be released in bottom-up (leaf-to-root) order. 8.5

that the multiple-granularity protocol requires that locks be acquired in top-down (root-to-leaf) order, whereas locks must be released in bottom-up (leaf to-root) order.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

159/202 **SUBMITTED TEXT** 32 WORDS **94%** **MATCHING TEXT** 32 WORDS

requires that each transaction T i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. 7.9

SA DBMS_AIML_FINAL.pdf (D111167082)

160/202 **SUBMITTED TEXT** 86 WORDS **97%** **MATCHING TEXT** 86 WORDS

A locking protocol is a set of rules that state when a transaction may lock and unlock each of the data items in the database. • The two-phase locking protocol allows a transaction to lock a new data item only if that transaction has not yet unlocked any data item. The protocol ensures serializability, but not deadlock freedom. In the absence of information concerning the manner in which data items are accessed, the two-phase locking protocol is both necessary and sufficient for ensuring serializability. •

A locking protocol is a set of rules that state when a transaction may lock and unlock each of the data items in the database. • Two-phase locking protocol: The two-phase locking protocol allows a transaction to lock a new data item only if that transaction has not yet unlocked any data item. The protocol ensures serializability, but not deadlock freedom. In the absence of information concerning the manner in which data items are accessed, the two-phase locking protocol is both necessary and sufficient for ensuring serializability. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

161/202	SUBMITTED TEXT	20 WORDS	91%	MATCHING TEXT	20 WORDS

Shared: If a transaction T i has obtained a shared-mode lock (denoted by S) on item Q, then

Shared. If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

162/202	SUBMITTED TEXT	27 WORDS	93%	MATCHING TEXT	27 WORDS

can read, but cannot write, Q. • Exclusive: If a transaction T i has obtained an exclusive-mode lock (denoted by X) on item Q, then

can read, but cannot write, Q. 2. Exclusive. If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

163/202	SUBMITTED TEXT	30 WORDS	100%	MATCHING TEXT	30 WORDS

Growing phase: A transaction may obtain locks, but may not release any lock. • Shrinking phase: A transaction may release locks, but may not obtain any new locks. •

Growing phase. A transaction may obtain locks, but may not release any lock. 2. Shrinking phase. A transaction may release locks, but may not obtain any new locks.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

164/202	SUBMITTED TEXT	35 WORDS	94%	MATCHING TEXT	35 WORDS

Lock point: The point in the schedule where the transaction has obtained 165 its final lock (the end of its growing phase) is called the lock point of the transaction. •

lock requests. The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the lock point of the transaction.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

165/202	SUBMITTED TEXT	67 WORDS	100%	MATCHING TEXT	67 WORDS

DDB) technology emerged as a merger of two technologies: (1) database technology, and (2) network and data communication technology. The latter has made tremendous strides in terms of wired and wireless technologies—from satellite and cellular communications and Metropolitan Area Networks (MANs) to the standardization of protocols like Ethernet, TCP/ IP, and the Asynchronous Transfer Mode (ATM) as well as the explosion of the Internet,

SA 47F417BA15858476660.pdf (D123781188)

166/202 **SUBMITTED TEXT** 94 WORDS **100%** **MATCHING TEXT** 94 WORDS

With advances in distributed processing and distributed computing that occurred in the operating systems arena, the database research community did considerable work to address the issues of data distribution, distributed query and transaction processing, distributed database metadata management, and other topics, and developed many research prototypes. However, a full-scale comprehensive DDBMS that 218 implements the functionality and techniques proposed in DDB research never emerged as a commercially viable product. Most major vendors redirected their efforts from developing a "pure" DDBMS product into developing systems based on client-server, or toward developing

SA 47F417BA15858476660.pdf (D123781188)

167/202 **SUBMITTED TEXT** 60 WORDS **97%** **MATCHING TEXT** 60 WORDS

We can define a distributed database (DDB) as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user. A collection of files stored at different nodes of a network and the maintenance of

SA 47F417BA15858476660.pdf (D123781188)

168/202 **SUBMITTED TEXT** 38 WORDS **100%** **MATCHING TEXT** 38 WORDS

among them via hyperlinks has become a common organization on the Internet, with files of Web pages. The common functions of database management, including uniform query processing and transaction processing, do not apply to this scenario yet.

SA 47F417BA15858476660.pdf (D123781188)

169/202 **SUBMITTED TEXT** 29 WORDS **100%** **MATCHING TEXT** 29 WORDS

Distributed Transactions Access to the various data items in a distributed system is usually accomplished through transactions, which must preserve the ACID properties. There are two types of

Distributed Transactions Access to the various data items in a distributed system is usually accomplished through transactions, which must preserve the ACID properties. There are two types of

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

170/202	SUBMITTED TEXT	35 WORDS	100%	MATCHING TEXT	35 WORDS

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases.

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

171/202	SUBMITTED TEXT	82 WORDS	97%	MATCHING TEXT	82 WORDS

Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site. The various transaction managers cooperate to execute global transactions. To understand how such a manager can be implemented, consider an abstract model of a transaction system, in which each site contains two subsystems: 237 • The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site. The various trans-action managers cooperate to execute global transactions. To understand how such a manager can be implemented, consider an abstract model of a transaction system, in which each site contains two subsystems: The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

172/202	SUBMITTED TEXT	64 WORDS	93%	MATCHING TEXT	64 WORDS

each such transaction may be either a local transaction (that is, a transaction that executes at only that site) or part of a global transaction (that is, a transaction that executes at several sites). • The transaction coordinator coordinates the execution of the various transactions (both local and global) initiated at that site. The overall system architecture appears in figure 12.1.

each such transaction may be either a local transaction (that is, a transaction that exe-cutes at only that site) or part of a global transaction (that is, a transaction that executes at several sites). The transaction coordinator coordinates the execution of the various trans-actions (both local and global) initiated at that site. The overall system architecture appears in Figure 1.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

173/202	SUBMITTED TEXT	175 WORDS	98%	MATCHING TEXT	175 WORDS

The structure of a transaction manager is similar in many respects to the structure of a centralized system. Each transaction manager is responsible for: • Maintaining a log for recovery purposes. • Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site. As we shall see, we need to modify both the recovery and concurrency schemes to accommodate the distribution of transactions. The transaction coordinator subsystem is not needed in the centralized environment, since a transaction accesses data at only a single site. A transaction coordinator, as its name implies, 238 is responsible for coordinating the execution of all the transactions initiated at that site. For each such transaction, the coordinator is responsible for: • Starting the execution of the transaction. • Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution. • Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

The structure of a transaction manager is similar in many respects to the structure of a centralized system. Each transaction manager is responsible for: 199 Maintaining a log for recovery purposes. Figure 1 System architecture. Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site. As we shall see, we need to modify both the recovery and concurrency schemes to accommodate the distribution of transactions. The transaction coordinator subsystem is not needed in the centralized en-vironment, since a transaction accesses data at only a single site. A transaction coordinator, as its name implies, is responsible for coordinating the execution of all the transactions initiated at that site. For each such transaction, the coordinator is responsible for: Starting the execution of the transaction. Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution. Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

174/202	SUBMITTED TEXT	92 WORDS	100%	MATCHING TEXT	92 WORDS

distributed system may suffer from the same types of failure that a centralized system does (for example, software errors, hardware errors, or disk crashes). There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are: • Failure of a site. • Loss of messages. • Failure of a communication link. • Network partition. The loss or corruption of messages is always a possibility in a distributed system. The system uses transmission-control protocols, such as TCP/ IP, to handle such errors.

Distributed system may suffer from the same types of failure that a centralized system does (for example, software errors, hardware errors, or disk crashes). There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are: Failure of a site. Loss of messages Failure of a communication link. Network partition. 200 The loss or corruption of messages is always a possibility in a distributed system. The system uses transmission-control protocols, such as TCP/IP, to handle such errors.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

175/202	SUBMITTED TEXT	125 WORDS	100%	MATCHING TEXT	125 WORDS

However, if two sites A and B are not directly connected, messages from one to the other must be routed through a sequence of communication links. If a communication link fails, messages that would have been transmitted across the link must be rerouted. In some cases, it is possible to find another route through the network, so that the messages are able to reach their destination. In other cases, a failure may result in there being no connection between some pairs of sites. A system is partitioned if it has been split into two (or more) subsystems, called partitions, that lack any connection between them. Note that, under this definition, a partition may consist of a single node. 239 12.3

However, if two sites A and B are not directly connected, messages from one to the other must be routed through a sequence of communication links. If a communication link fails, messages that would have been transmitted across the link must be rerouted. In some cases, it is possible to find another route through the network, so that the messages are able to reach their destination. In other cases, a failure may result in there being no connection between some pairs of sites. A system is partitioned if it has been split into two (or more) subsystems, called partitions, that lack any connection between them. Note that, under this definition, a partition may consist of a single node.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

176/202	SUBMITTED TEXT	305 WORDS	95%	MATCHING TEXT	305 WORDS

Directory systems Consider an organization that wishes to make data about its employees available to a variety of people in the organization; examples of the kinds of data include name, designation, employee-id, address, email address, phone number, fax 246 number, and so on. In the pre-computerization days, organizations would create physical directories of employees and distribute them across the organization. Even today, telephone companies create physical directories of customers. In general, a directory is a listing of information about some class of objects such as persons. Directories can be used to find information about a specific object, or in the reverse direction to find objects that meet a certain requirement. In the world of physical telephone directories, directories that satisfy lookups in the forward direction are called white pages, while directories that satisfy lookups in the reverse direction are called yellow pages. In today's networked world, the need for directories is still present and, if anything, even more important. However, directories today need to be available over a computer network, rather than in a physical (paper) form. Directory Access Protocols Directory information can be made available through Web interfaces, as many organizations, and phone companies in particular, do. Such interfaces are good for humans. However, programs too need to access directory information. Directories can be used for storing other types of information, much like file system directories. For instance, Web browsers can store personal bookmarks and other browser settings in a directory system. A user can thus access the same settings from multiple locations, such as at home and at work, without having to share a file system. Several directory access protocols have been developed to provide a standardized way of accessing data in a directory. The most widely used among them today is the Lightweight Directory Access Protocol (LDAP).

Directory Systems Consider an organization that wishes to make data about its employees avail-able to a variety of people in the organization; examples of the kinds of data include name, designation, employee-id, address, email address, phone number, fax number, and so on. In the precomputerization days, organizations would cre-ate physical directories of employees and distribute them across the organization. Even today, telephone companies create physical directories of customers. In general, a directory is a listing of information about some class of objects such as persons. Directories can be used to find information about a specific object, or in the reverse direction to find objects that meet a certain requirement. In the world of physical telephone directories, directories that satisfy lookups in the forward direction are called white pages, while directories that satisfy lookups in the reverse direction are called yellow pages. In today's networked world, the need for directories is still present and, if anything, even more important. However, directories today need to be available over a computer network, rather than in a physical (paper) form. 1. Directory Access Protocols Directory information can be made available through Web interfaces, as many organizations, and phone companies in particular, do. Such interfaces are good for humans. However, programs too need to access directory information. Direc-tories can be used for storing other types of information, much like file system directories. For instance, Web browsers can store personal bookmarks and other browser settings in a directory system. A user can thus access the same settings from multiple locations, such as at home and at work, without having to share a file system. 225 Several directory access protocols have been developed to provide a stan-dardized way of accessing data in a directory. The most widely used among them today is the Lightweight Directory Access Protocol (LDAP).

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

177/202 **SUBMITTED TEXT** 68 WORDS **100%** **MATCHING TEXT** 68 WORDS

Distributed databases bring the advantages of distributed computing to the database management domain. A distributed computing system consists of a number of processing elements, not necessarily homogeneous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks. As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it efficiently in a coordinated manner. 11.3

SA 47F417BA15858476660.pdf (D123781188)

178/202 **SUBMITTED TEXT** 122 WORDS **98%** **MATCHING TEXT** 122 WORDS

In general a directory system is implemented as one or more servers, which service multiple clients. Clients use the application programmer interface defined by the directory system to communicate with the directory servers. Directory access protocols also define a data model and access control. 247 The X.500 directory access protocol, defined by the International Organization for Standardization (ISO), is a standard for accessing directory information. However, the protocol is rather complex, and is not widely used. The Lightweight Directory Access Protocol (LDAP) provides many of the X.500 features, but with less complexity, and is widely used. In the rest of this section, we shall outline the data model and access protocol details of LDAP . 12.5

In general a directory system is implemented as one or more servers, which service multiple clients. Clients use the application programmer interface defined by the directory system to communicate with the directory servers. Directory access protocols also define a data model and access control. The X.500 directory access protocol, defined by the International Organiza-tion for Standardization (ISO), is a standard for accessing directory information. However, the protocol is 226 rather complex, and is not widely used. The Lightweight Directory Access Protocol (LDAP) provides many of the X.500 features, but with less complexity, and is widely used. In the rest of this section, we shall outline the data model and access protocol details of LDAP. 2.1.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

179/202 **SUBMITTED TEXT** 91 WORDS **97%** **MATCHING TEXT** 91 WORDS

One of the goals in using distributed databases is high availability; that is, the database must function almost all the time. In particular, since failures are more likely in large distributed systems, a distributed database must continue functioning even when there are various types of failures. The ability to continue functioning even during failures is referred to as robustness. For a distributed system to be robust, it must detect failures, reconfigure the system so that computation may continue, and recover when a processor or a link is repaired. 12.6

One of the goals in using distributed databases is high availability; that is, the database must function almost all the time. In particular, since failures are more likely in large distributed systems, a 211 distributed database must continue func-tioning even when there are various types of failures. The ability to continue functioning even during failures is referred to as robustness. For a distributed system to be robust, it must detect failures, reconfigure the system so that computation may continue, and recover when a processor or a link is repaired.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

180/202	SUBMITTED TEXT	37 WORDS	100%	MATCHING TEXT	37 WORDS

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases. ● The

that we need to consider. The local transactions are those that access and update data in only one local database; the global transactions are those that access and update data in several local databases. The

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

181/202	SUBMITTED TEXT	20 WORDS	100%	MATCHING TEXT	20 WORDS

The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

182/202	SUBMITTED TEXT	17 WORDS	90%	MATCHING TEXT	17 WORDS

The ability to continue functioning even during failures is referred to as robustness. ● Snapshot: A

The ability to continue functioning even during failures is referred to as robustness. a

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

183/202	SUBMITTED TEXT	13 WORDS	83%	MATCHING TEXT	13 WORDS

data models such as the network model or the hierarchical model. The

data models, such as the network model (see Appendix D) or the hierarchical model . the

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

184/202	SUBMITTED TEXT	103 WORDS	99%	MATCHING TEXT	103 WORDS

Design The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data. The needs of the users play a central role in the design process. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broad set of issues. These additional aspects of the expected use of the database influence a variety of design choices at the physical, logical, and view levels. Phases

Design Process The task of creating a database application is a complex one, involving • design of the database schema • design of the programs that access and update the data and • Design of a security scheme to control access to data. The needs of the users play a central role in the design process. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broad set of issues. These additional aspects of the expected use of the database influence a variety of design choices at the physical, logical, and view levels. Design Phases

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

185/202	SUBMITTED TEXT	88 WORDS	97%	MATCHING TEXT	88 WORDS

A high-level data model serves the database designer by providing a conceptual 267 framework in which to specify, in a systematic fashion, the data requirements of the database users, and a database structure that fulfills these requirements. 1. The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

A high-level data model serves the database designer by providing a conceptual framework in which to specify, in a systematic fashion, the data requirements of the database users, and a database structure that fulfills these requirements. • The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact ex-tensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements. •

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

186/202	SUBMITTED TEXT	47 WORDS	94%	MATCHING TEXT	47 WORDS

of user requirements. 2. Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The

of user requirements. • Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase pro-vides a detailed overview of the enterprise. Typically, the

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

187/202	SUBMITTED TEXT	197 WORDS	97%	MATCHING TEXT	197 WORDS

Typically, the conceptual- design phase results in the creation of an entity-relationship diagram that provides a graphic representation of the schema. 3. A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements. 4. The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. ● In the logical-design phase, the designer maps the high-level conceptual 268 schema onto the implementation data model of the database system that will be used. The implementation data model is typically the relational data model, and this step typically consists of mapping the conceptual schema defined using the entity-relationship model into a relation schema. ● Finally, the designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified.

Typically, the conceptual-design phase re-sults in the creation of an entity- relationship diagram that provides a graphic representation of the schema. • A fully developed conceptual schema also indicates the functional require-ments of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements. The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. o In the logical-design phase, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The implementation data model is typically the relational data model, and this step typically consists of mapping the conceptual schema defined using the entity- relationship model into a relation schema. o Finally, the designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

188/202	SUBMITTED TEXT	26 WORDS	55%	MATCHING TEXT	26 WORDS

normal forms are as follows: o First normal form o Second normal form o Third normal form o Boyce-Codd normal form o Fifth normal form

normal forms are there? They First Normal Form Second Normal Form Third Normal Form Boyce-Codd Normal Form Fourth Form Fifth Normal Form

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

189/202	SUBMITTED TEXT	38 WORDS	100%	MATCHING TEXT	38 WORDS

normalization. When we define an E-R diagram carefully, identifying all entities correctly, the relation schemas generated from the E-R diagram should not need much further normalization. However, there can be functional dependencies between attributes of an entity.

Normalization When we define an E-R diagram carefully, identifying all entities correctly, the relation schemas generated from the E-R diagram should not need much further normalization. However, there can be functional dependencies between attributes of an entity.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

190/202	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

Most examples of such dependencies arise out of poor E-R diagram design.

Most examples of such dependencies arise out of poor E-R diagram design.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

191/202	SUBMITTED TEXT	37 WORDS	100%	MATCHING TEXT	37 WORDS

Similarly, a relationship set involving more than two entity sets may result in a schema that may not be in a desirable normal form. Since most relationship sets are binary, such cases are relatively rare.

Similarly, a relationship set involving more than two entity sets may result in a schema that may not be in a desirable normal form. Since most relationship sets are binary, such cases are relatively rare. (

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

192/202	SUBMITTED TEXT	71 WORDS	100%	MATCHING TEXT	71 WORDS

dependencies can help us detect poor E-R design. If the generated 270 relation schemas are not in desired normal form, the problem can be fixed in the E-R diagram. That is, normalization can be done formally as part of data modeling. Alternatively, normalization can be left to the designer's intuition during E-R modeling, and can be done formally on the relation schemas generated from the E-R model.

dependencies can help us detect poor E-R design. If the generated relation schemas are not in desired normal form, the problem can be fixed in the E-R diagram. That is, normalization can be done formally as part of data modeling. Alternatively, normalization can be left to the designer's intuition during E-R modeling, and can be done formally on the relation schemas generated from the E-R model.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

193/202	SUBMITTED TEXT	132 WORDS	100%	MATCHING TEXT	132 WORDS

If a multivalued dependency holds and is not implied by the corresponding functional dependency, it usually arises from one of the following sources: • A many-to-many relationship set. • A multivalued attribute of an entity set. For a many-to-many relationship set each related entity set has its own schema and there is an additional schema for the relationship set. For a multivalued attribute, a separate schema is created consisting of that attribute and the primary key of the entity set (as in the case of the phone number attribute of the entity set instructor). The universal-relation approach to relational database design starts with an assumption that there is one single relation schema containing all attributes of interest. This single schema defines how users and applications interact with the database. 13.4

If a multivalued dependency holds and is not implied by the corresponding functional dependency, it usually arises from one of the following sources: A many-to-many relationship set. A multivalued attribute of an entity set. 141 For a many-to-many relationship set each related entity set has its own schema and there is an additional schema for the relationship set. For a multivalued attribute, a separate schema is created consisting of that attribute and the primary key of the entity set (as in the case of the phone number attribute of the entity set instructor). The universal-relation approach to relational database design starts with an assumption that there is one single relation schema containing all attributes of interest. This single schema defines how users and applications interact with the database.

W https://annamalaiuniversity.ac.in/studport/download/sci/cis/resources/MCA01-%20RDBMS%20-%2019MCAC ...

194/202 **SUBMITTED TEXT** 52 WORDS **86%** **MATCHING TEXT** 52 WORDS

the values in each row. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is called a domain.

SA DBMS_AIML_FINAL.pdf (D111167082)

195/202 **SUBMITTED TEXT** 46 WORDS **95%** **MATCHING TEXT** 46 WORDS

In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is called a domain. •

SA DBMS_AIML_FINAL.pdf (D111167082)

196/202 **SUBMITTED TEXT** 122 WORDS **100%** **MATCHING TEXT** 122 WORDS

ARIES uses a number of techniques to reduce the time taken for recovery, and to reduce the overhead of checkpointing. In particular, ARIES is able to avoid redoing many logged operations that have already been applied and to reduce the amount of information logged. The price paid is greater complexity; the benefits are worth the price. The major differences between ARIES and the recovery algorithm presented earlier are that ARIES: 1. Uses a log sequence number (LSN) to identify log records, and stores LSNs in database pages to identify which operations have been applied to a database page. 2. Supports physiological redo operations, which are physical in that the affected page is physically identified, but can be logical within the page.

ARIES uses a number of techniques to reduce the time taken for recovery, and to reduce the overhead of checkpointing. In particular, ARIES is able to avoid redoing many logged operations that have already been applied and to reduce the amount of information logged. The price paid is greater complexity; the benefits are worth the price. The major differences between ARIES and the recovery algorithm presented earlier are that ARIES: 1. Uses a log sequence number (LSN) to identify log records, and stores LSNs in database pages to identify which operations have been applied to a database page. 2. Supports physiological redo operations, which are physical in that the affected page is physically identified, but can be logical within the page. 3.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

197/202 **SUBMITTED TEXT** 77 WORDS **100%** **MATCHING TEXT** 77 WORDS

Uses a dirty page table to minimize unnecessary redos during recovery. As mentioned earlier, dirty pages are those that have been updated in memory, and the disk version is not up-to-date. 348 4. Uses a fuzzy-checkpointing scheme that records only information about dirty pages and associated information and does not even require writing of dirty pages to disk. It flushes dirty pages in the background, continuously, instead of writing them during checkpoints.

Uses a dirty page table to minimize unnecessary redos during recovery. As mentioned earlier, dirty pages are those that have been updated in memory, and the disk version is not up-to-date. 4. Uses a fuzzy-checkpointing scheme that records only information about dirty pages and associated information and does not even require writing of dirty pages to disk. It flushes dirty pages in the background, continuously, instead of writing them during checkpoints. 4.22.2

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

198/202	SUBMITTED TEXT	126 WORDS	98%	MATCHING TEXT	126 WORDS

Recovery Algorithm ARIES recovers from a system crash in three passes: • Analysis pass: This pass determines which transactions to undo,which pages were dirty at the time of the crash, and the LSN from which the redo pass should start. • Redo pass: This pass starts from a position determined during analysis, and performs a redo, repeating history, to bring the database to a state it was in before the crash. • Undo pass: This pass rollback all transactions that were incomplete at the time of crash. 350 • Other Features Among other key features that ARIES provides are: • Nested top actions: ARIES allows the logging of operations that should not be undone even if a transaction gets rolled back;

Recovery Algorithm: ARIES recovers from a system crash in three passes. • Analysis pass: This pass determines which transactions to undo, which pages were dirty at the time of the crash, and the LSN from which the redo pass should start. • Redo pass: This pass starts from a position determined during analysis, and performs a redo, repeating history, to bring the database to a state it was in before the crash. • Undo pass: This pass rolls back all transactions that were incomplete at the time of crash. 4.22.3 Other Features: Among other key features that ARIES provides are: • Nested top actions: ARIES allows the logging of operations that should not be undone even if a transaction gets rolled back.

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

199/202	SUBMITTED TEXT	13 WORDS	100%	MATCHING TEXT	13 WORDS

Such operations that should not be undone are called nested top actions.

Such operations that should not be undone are called nested top actions. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

200/202	SUBMITTED TEXT	102 WORDS	100%	MATCHING TEXT	102 WORDS

Recovery independence: Some pages can be recovered independently from others, so that they can be used even while other pages are being recovered. If some pages of a disk fail, they can be recovered without stopping transaction processing on other pages. • Savepoints: Transactions can record savepoints, and can be rolled back partially, up to a savepoint. This can be quite useful for deadlock handling, since transactions can be rolled back up to a point that permits release of required locks, and then restarted from that point. Programmers can also use savepoints to undo a transaction partially, and then continue execution;

Recovery independence: Some pages can be recovered independently from others, so that they can be used even while other pages are being recovered. If some pages of a disk fail, they can be recovered without stopping transaction processing on other pages. • Savepoints: Transactions can record savepoints, and can be rolled back partially, up to a savepoint. This can be quite useful for deadlock handling, since transactions can be rolled back up to a point that permits release of required locks, and then restarted from that point. Programmers can also use savepoints to undo a transaction partially, and then continue execution. •

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

201/202	SUBMITTED TEXT	101 WORDS	100%	MATCHING TEXT	101 WORDS

execution. • Fine-grained locking: The ARIES recovery algorithm can be used with index concurrency-control algorithms that permit tuple-level locking on indices, instead of page-level locking, which improves concurrency significantly. • Recovery optimizations: The DirtyPageTable can be used to prefetch pages during redo, instead of fetching a page only when the system finds a log record to be applied to the page. Out-of-order redo is also possible: Redo 351 can be postponed on a page being fetched from disk, and performed when the page is fetched. Meanwhile, other log records can continue to be processed. •

execution. • Fine-grained locking: The ARIES recovery algorithm can be used with index concurrency-control algorithms that permit tuple-level locking on indices, instead of page-level locking, which improves concurrency significantly. • Recovery optimizations: The DirtyPageTable can be used to prefetch pages during redo, instead of fetching a page only when the system finds a log record to be applied to the page. Out-of-order redo is also possible: Redo can be postponed on a page being fetched from disk, and performed when the page is fetched. Meanwhile, other log records can continue to be processed. 4.23

W http://www.gpcet.ac.in/wp-content/uploads/2018/08/DBMS-NOTESR15-Unit-46-ok.doc

202/202	SUBMITTED TEXT	15 WORDS	95%	MATCHING TEXT	15 WORDS

A row is called a _____ , a column header is called an _____.

SA DBMS_AIML_FINAL.pdf (D111167082)