SPPU New Syllabus

A Book Of

ADVANCED DBMS

For MCA (Management): Semester - II

[Course Code – IT24: Credit - 3]

CBCS Pattern

As Per New Syllabus, Effective from June 2020

Prof. Gautam Bapat

B.Sc., M.C.A., P.G.D.B.M., M.M.S
Assistant Professor (Computer Applications & Marketing)
Program Head - BBA (Global eBusiness)
School of Management (UG Programme)
MIT World Peace University, Pune

Price ₹ 260.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N5320

ADVANCED DBMS (MCA-MANAGEMENT) (SEM-II) **ISBN 978-93-5451-109-7**

First Edition : **March 2021**

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
STAR COPIERS PVT. LTD.
Kumthekar Road, Sadashiv Peth,
PUNE - 411 030
Tel - (020) 24479201

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
(For orders within Pune) Tel : (020) 2445 2044, Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan : S. No. 28/27, Dhayari, Near Asian College Pune 411041
(For orders outside Pune) Tel : (020) 24690204 Fax : (020) 24690316; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com/mumbai@niralibooks.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : jalgaon@niralibooks.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : kolhapur@niralibooks.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : nagpur@niralibooks.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : delhi@niralibooks.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: bengaluru@niralibooks.com

Other Branches : Hyderabad, Chennai

Preface ...

We take an opportunity to present this Text Book on **"Advanced DBMS"** to the students of MCA (Management), Semester-II as per New Syllabus (CBCS pattern) 2020.

The objective of this book is to present the subject matter in a most concise, compact, to the point and lucid manner.

This book will help the readers to have a broader view on Advanced concepts of Database Management System. The language used in this book is easy and will help students to improve their vocabulary of Technical terms and understand the matter in a better and happier way.

I sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in me and giving me this opportunity to reach out to the students of management studies.

Any suggestions for improving the content shall be gratefully received and highly appreciated.

Author

Syllabus ...

1. Introduction DBMS – Concepts & Architectures (Weightage-10%) (Sessions-4)

1.1 Database and Need for DBMS, Characteristics of DBMS

1.2 Database 3-tier schema (ANSI/SPARC) and system architecture of DBMS

1.3 Views of data- Schemas and instances, Data Independence

1.4 Centralized, Client-Server system, Transaction servers, Data servers, Cloud based servers

Extra Reading: Indexing and Hashing - Basic concepts of indexing, ordered index, B+ tree index, B+ tree extensions, Multiple key access, Hashing concepts, types of hashing, Bitmap indices

2. Data Modelling and Relational Database Design (Weightage-16%) (Sessions-8)

2.1 Data Modelling using ER Diagram: Representation of Entities, Attributes, Relationships and their Type, Cardinality, Generalization, Specialization, Aggregation.

2.2 Relational data model: Structure of Relational Database Model, Types of keys, Referential Integrity Constraints

2.3 Codd's rules

2.4 Database Design using E-R, E-R to Relational

2.5 Normalization – Normal forms based on primary (1NF, 2NF, 3NF, BCNF)

Note: Case studies based on E-R diagram & Normalization

Extra Reading: Database languages - Relational Algebra, Relational database languages, Data definition in SQL, Views and Queries in SQL, Joins, specifying constraints and Indexes in SQL, Specifying constraints management systems Postgres/ SQL/MySQL

3. Transaction and Concurrency Control (Weightage-13%) (Sessions-6)

3.1 Concept of transaction, ACID properties, States of transaction

3.2 Concurrency control, Problems in concurrency controls

3.3 Scheduling of transactions, Serializability and testing of serializability

3.4 Lock-based Protocol and Time stamp-based ordering protocols

3.5 Deadlock Handling

Extra Readings: Semantic data controls & Multi-version concurrency control

4. Parallel Databases (Weightage-13%) (Sessions-6)

4.1 Introduction to Parallel Databases

4.2 Parallel Database Architectures

4.3 I/O parallelism

4.4 Inter-query and Intra-query parallelism

4.5 Inter-operational and Intra-operational parallelism

4.6 Key elements of parallel database processing: Speed-up, Scale-up Synchronization and Locking

Extra Readings: Parallel handling and Load balancing

5. Distributed Databases **(Weightage-13%) (Sessions-6)**

5.1 Introduction to Distributed Database System

5.2 Homogeneous and Heterogeneous Databases

5.3 Distributed data storage (Fragmentation and Replication)

5.4 Distributed transactions

5.5 Concurrency control schemes in DDBMS

5.6 Commit protocols 2 phase and 3 Phase Commit Protocol

Extra Readings: Reliability issues in DDBMS and Web based interface of DDBMS

6. Object Oriented Databases & Applications **(Weightage-10%) (Sessions-4)**

6.1 Overview of Object- Oriented Database concepts & characteristics

6.2 Database design for OODBMS – Objects, OIDs and reference type

6.3 Spatial data and Spatial indexing (Any two techniques)

6.4 Mobile Database: Need, Structure, Features, Limitations and Applications

6.5 Temporal databases, temporal aspects valid time, transaction time or decision time

6.6 Multimedia Database: Architecture, Type and Characteristics

7. Crash Recovery and Backup **(Weightage-10%) (Sessions-5)**

7.1 Failure classifications

7.2 Recovery & Atomicity

7.3 Log based recovery

7.4 Checkpoint and Shadow Paging in Data recovery

7.5 Database backup and types of backups

Extra Readings: Role and Functions of Database administrator

8. Security and Privacy **(Weightage-10%) (Sessions-4)**

8.1 Database security issues

8.2 Discretionary access control based on grant & revoking privilege

8.3 Mandatory access control and role-based access control for multilevel security

8.4 Encryption & public key infrastructures

9. NO-SQL Database **(Weightage-5%) (Sessions-2)**

Introduction, Types of NoSQL, Need of NoSQL databases, Use Cases

❖❖❖

Contents ...

❖❖❖

1... Introduction DBMS – Concepts and Architectures

Objectives...

- To understand core concepts of Database Management System.
- To get information of database 3-tier schema and system architecture of DBMS.
- To study various views of data- Schemas, instances and data independence.
- To know about Centralized, Client-Server system, Transaction servers, Data servers, Cloud based servers.

1.1 INTRODUCTION

- Database is a collection of data. Database contains information about one particular enterprise.
  For Example,
  - Bank which stores customers banking data.
  - Hospital which stores patient data.
  - University which stores student data.
- The objective of DBMS is to provide convenient and effective method of defining, storing and retrieving the information contained in the database.
- In addition, the DBMS must provide for the safety of the stored information. It should protect the data from system crash or attempt at unauthorized access. If the data are to be shared among several users, the system must avoid possible anomalous results.
- **Examples of DBMS Software:**
  Oracle, Microsoft Access, FoxPro, MongoDB.

1.1.1 Definition of DBMS

- A database management system is a computer based system or program to record and maintain information or data.
- In any organization, data is the basic resource needed to run the organization. This data is required by decision makers for processing and retrieving information.
- A data is collection of information or real fact which can be recorded and have implicit meaning. Customer_name, item_price, balance etc. can be considering as data.
- The database is used to store information useful to an organization.
- A database is a shared collection of interrelated data which is designed to fulfill the information needs.

  For example: Consider the names, telephone numbers and addresses of the people you know. This data is recorded in an indexed address book or stored on a diskette or using a personal computer. This is a collection of related data with an implicit meaning and hence is a database.
- A database uses the following implicit properties:
  1. A database represents some aspect of the real world, sometimes called the miniworld. Changes to the miniworld are reflected in the database.
  2. A database is a collection of data with some meaning. A random selection of data cannot correctly be referred to as a database.
  3. A database is designed, built and populated with data for a specific purpose. It has an intended group of users.
- So, finally we can say that a database has some source from which data are derived, some degree of interaction with events in the real world and an audience that is actively interested in the contents of the database.
- A Database Management System (DBMS) is a software system that allows user to define, manipulate and process the data in a database, in order to produce meaningful information.
- The basic functions of DBMS are:
  1. To store data in a database.
  2. To organize the data.
  3. To control access of data.
  4. To protect data i.e. provide security.
- Hence, the DBMS is a general purpose software system that facilitates the processes of *defining*, *constructing* and *manipulating* databases for various applications.
  1. **Defining a database** involves specifying the data types, structures and constraints for the data to be stored in the database.

2. **Constructing the database** is the process of storing the data itself on some storage medium that is controlled by the DBMS.
3. **Manipulating a database** includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld and generating reports from the data. So in general, user can write programs or queries; DBMS use the database stored on storage devices and gives meaningful information.

1.1.2 Components and Overall Structure of DBMS

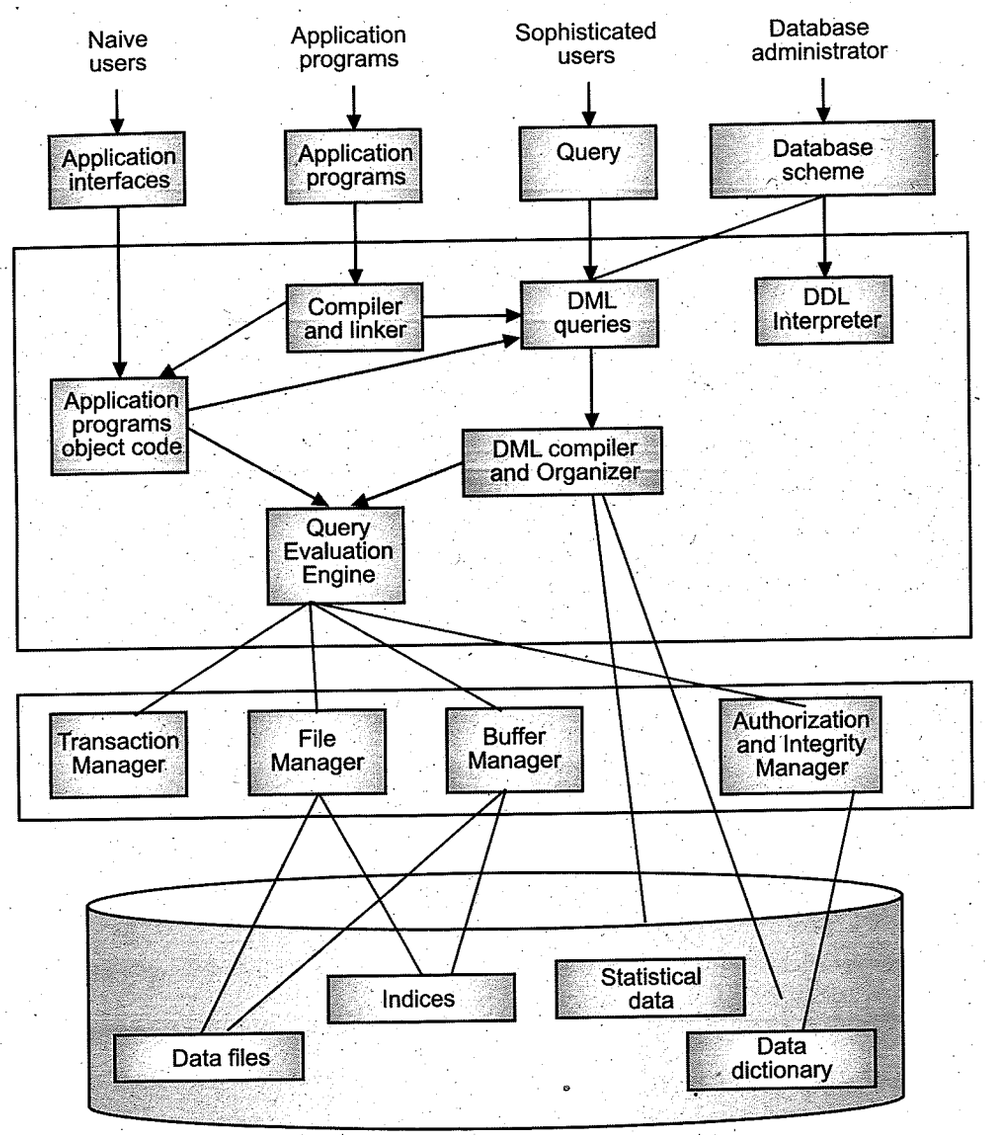


Fig. 1.1: DBMS System Structure

- Components of DBMS are broadly classified as follows:

1. **Query Processor Components:**
   - (i) **DML Pre-compiler:** It translates DML statements in a query language into low level instructions that query evaluation engine understand. It also attempts to transform user's request into an equivalent but more efficient form.
   - (ii) **Embedded DML Pre-compiler:** It converts DML statements embedded in an application program to normal procedure calls in the host language. The Pre-compiler must interact with the DML compiler to generate the appropriate code.
   - (iii) **DDL Interpreter:** It interprets the DDL statements and records them in a set of tables containing metadata or data dictionary.
   - (iv) **Query Evaluation Engine:** It executes low-level instructions generated by the DML compiler.
2. **Storage Manager Components:** They provide the interface between the low-level data stored in the database and application programs and queries submitted to the system.
   - (i) **Authorization and Integrity Manager:** It tests for the satisfaction of integrity constraints checks the authority of users to access data.
   - (ii) **Transaction Manager:** It ensures that the database remains in a consistent state despite the system failures and that concurrent transaction execution proceeds without conflicting.
   - (iii) **File Manager:** It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
   - (iv) **Buffer Manager:** It is responsible for fetching data from disk storage into main memory and deciding what data to cache in memory.
3. **Data Structures:** Following data structures are required as a part of the physical system implementation:
   - (i) **Data Files:** It stores the database.
   - (ii) **Data Dictionary:** It stores metadata (information about data) about the structure of the database.
   - (iii) **Indices:** Provides fast access to data items that hold particular values.
   - (iv) **Statistical Data:** It stores statistical information about the data in the database. This information is used by query processor to select efficient ways to execute query.

1.1.3 Functions of DBMS

- DBMS performs following functions:
  1. **Data Security and Data Integrity:** This is the most important function of database which handles the security and integrity scheme of database application.

  2. **Data Definition:** Data definition defines the structure of database. It is also defines field size, record structure.
  3. **Data Manipulation:** It contains manipulation of data, i.e. inserting, modifying, and deleting function.
  4. **Data Recovery:** After system failure to recover data in the database.
  5. **Concurrency:** To handle concurrent access of multiple users.

1.1.4 Database Languages

- Data language consists of two parts:
  1. **Data Definition Language:** To specify the database schema.
  2. **Data Manipulation Language:** To express database queries and updates.

1. **Data Definition Language:**

- Database schema is specified by a set of definitions which are expressed by a special language called Data Definition Language (DDL).
  - (a) **Data Dictionary:** The result of compilation of DDL statements is a set of tables which is stored in a special file called Data Dictionary or system catalogues. This file contain metadata i.e. data about data.
  - (b) **Data Storage and Definition Language:** The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called Data Storage and Definition Language.

2. **Data Manipulation Language:**

- Data Manipulation Language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.
- Data manipulation means:
  - To retrieve the information from database.
  - To insert information into database.
  - To delete information from database.
  - To modify information from database.
- There are two types of DML:
  1. **Procedural DML:** It requires a user to specify what data are needed and how to get those data.
  2. **Non-procedural DML:** It requires a user to specify what data are needed without specifying how to get those data.

1.1.5 Advantages of DBMS

Following are the advantages of DBMS:

1. **Centralized Management and Control over Data:** Database administrator is the person having central control over the system.

2. **Reduction of redundancies:** Centralized control of data by DBA avoids unnecessary duplication of data.
3. **Shared Data:** DBMS allows the sharing of data under its control by any number of application programmers and users.
4. **Integrity:** Centralized control also ensures that adequate checks are incorporated in the database to provide data integrity.
5. **Security:** The DBA can ensure that proper access procedures are followed, including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data.
6. **Conflict Resolution:** DBA resolves the conflicting requirements of various users and applications.
7. **Data Independence:** DBA can modify the structure of data record. This modification does not affect other applications.

1.1.6 Disadvantages of DBMS

Disadvantages of DBMS are given below:

1. Number of problems has associated with centralization of data.
2. Cost of software and hardware is high.
3. Complexity of back-up and recovery is increased.
4. DBMS is not designed for smaller database.
5. Problems occur due to improper database design.

1.1.7 Applications of DBMS

- DBMS used everywhere, where large amount of data processed. Some applications of DBMS are as given below.

1. **Universities:** To store data related colleges, courses, grades, number of seats etc.
2. **Banking:** To store bank data such as customer account, loan, staff and the most important is day to day bank transactions.
3. **Scientific Applications:** Database is used scientific applications such as space observation, forecasting etc.
4. **Telecommunication:** Database is useful for storing information of calls made, numbers of messages send and received, balance of prepaid card and for network details in Telecommunication.
5. **Finance:** Database stores information such as shares and other financial data. DBMS also stores information of mutual funds, shares, bonds etc.
6. **Marketing:** It stores the information such as marketing plan, purchase, products, sales etc.
7. **Government:** Most government departments are now computerized and they store their interrelated data in database.

1.2 DATABASE AND NEED FOR DBMS, CHARACTERISTICS OF DBMS

- We know that a database is a collection of interrelated data of an organization or enterprise. This data is stored and manipulated using DBMS.
- A file system is an organization of storage space designed to contain files in directories. Before the evolution of DBMS, organizations used to store information in file systems.
- A typical file processing system is supported by a conventional operating system. The system stores permanent records in various files and requires application programs to extract records, or to add or delete records.
- File system has various drawbacks such as Data isolation, data inconsistency, security problem etc. We need DBMS to avoid these problems.

1.2.1 Need of DBMS

Following are some points regarding to need of DBMS.

1. **To avoid data redundancy and inconsistency:**

- Since the data files and application programs are created by different programmers over a long period,
  - (i) The data files are likely to have different formats.
  - (ii) Programs may be written in several programming languages.
  - (iii) The same information may be duplicated in several plates. This results in data redundancy and inconsistency.
- For example, consider following two data files:
  - (i) ***Savings account* data file:** Stores information about customer {account_no, name, social_security, address, telephone_no}.
  - (ii) ***Checkings account* data file:** Stores information about customer {account_no, name, social_security, address, telephone_no}.
- Fields {name, social_security, address and telephone_no} are same in both the files i.e. duplication of data results in data redundancy. Data redundancy increases the cost of storing and retrieving the data. If the values of these common fields are not matching for some records in both files, then it results in inconsistency of data.
- A DBMS uses *data normalization* to avoid redundancy and duplicates.

2. **For easy access of data:**

- Conventional file processing system does not allow needed data to be retrieved in a convenient and efficient manner.
- **For example:** Consider a data file 'Savings account' with fields {acc_no, name, social_security, address, balance}. Application programs to access the data are written. But if user wants to display only those records for which balance is greater than ₹ 10,000. If that program is not written, then it is difficult to access that data.

3. **For data isolation:**

- Because data are scattered in various files and file may be in different formats, it is difficult to write new application programs to retrieve the appropriate data.

4. **To solve integrity problems:**

- The data values stored in the database must satisfy certain types of consistency constraints. Application programmers enforce these consistency constraints by adding appropriate code in the various application programs. However, when a new constraint is to be added, it is difficult to change the programs to enforce the new constraint.
- Data integrity ensures that only required data is stored in the database. Data is validated before entered into the database using integrity constraints such as primary key, foreign key, etc.

5. **To solve atomicity problems:**

- A computer system is subject to failure. In many applications it is crucial to ensure that once a failure has occurred and has been detected, the data are stored to the consistent state that existed prior to the failure. It is difficult to ensure this property in a conventional file-processing system.

  For example: Consider a program to transfer ₹ 500 from account A to account B.
- If failure occurs after removing ₹ 500 from account A and before adding ₹ 500 to account B. This results in inconsistent state.

6. **To solve security problems:**

- DBMS allows organizations to enforce policies that enable compliance and security. DBMS is also responsible to maintain optimum performance of querying operations.

1.2.2 Characteristics of DBMS

Characteristics of DBMS are given below:

1. **File processing approach of users:** In file processing, each user defines and implements the files needed for a specific application, therefore more storage space is required. In DBMS, a single database is maintained that is defined once and then is accessed by various users. DBMS software is not written for specific application.
2. **Self-describing nature of DBMS system:** A DBMS contains database as well as a complete definition of the database, which is stored in the system catalog. It contains information such as structure of each file, the type and storage format of each data item and various constraints of the data. The catalog is used by database users who need information about database.

   For example: A company database, a banking database, a university database.

   In file processing, data definition is part of the application programs. These programs works with only one specific database, unlike DBMS can access diverse databases by extracting the database definitions from the catalog.

3. **Isolation between programs and data:** DBMS access programs are written independently of any specific files. The structure of data files is stored in catalog separately from the access program (program-data independence). In file processing, if any change in structure of file is made then all programs that access this file have to be changed. Suppose we want to add any field in record of file (say birth-date in student file) then program has to be change in file processing.
4. **Multiple views:** DBMS supports multiple views of the data which file cannot support. For example: one user is only interested for student mark-list while other user is interested for courses attended by that student; these multi-user views are satisfied by DBMS.
5. **Sharing of data** is possible only in DBMS, not in files.

1.3 DATABASE 3-TIER SCHEMA (ANSI/SPARC) AND SYSTEM ARCHITECTURE OF DBMS

- DBMS architecture is the way in which the data in a database is viewed (or represented to) by users. It helps you represent your data in an understandable way to the users, by hiding the complex bits that deal with the working of the system.
- The ANSI-SPARC Architecture, where ANSI-SPARC stands for American National Standards Institute, Standards Planning And Requirements Committee, is an abstract design standard for a Database Management System (DBMS), first proposed in 1975.
- The ANSI-SPARC database architecture is the basis of most of the modern databases.
- The three levels present in this architecture are Physical level, Conceptual level and External level.
- Fig. 1.2 shows the three levels of system architecture.

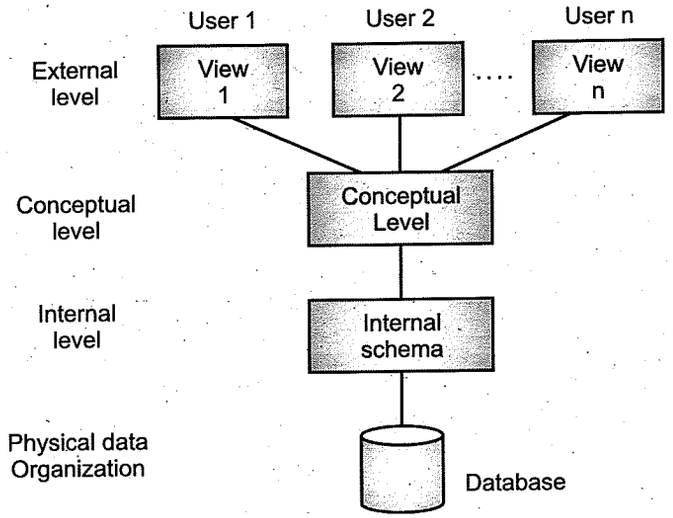


Fig. 1.2: The ANSI-SPARC three level/3-tier architecture of Database.

1. **Physical Level:** This is the lowest level in the three level architecture. It is also known as the Internal level. It describes how the data is actually stored. It also describes the data structures and access methods to be used by the database. At the physical level, complex low-level data structures are described in detail. The internal view is expressed by the internal schema which contains the definition of stored record, the method of representing the data fields and the access aids used.
2. **Conceptual Level:** The conceptual level is at a higher level than the physical level. It is also known as the logical level. It describes *what* data are actually stored in the database and the relationship that exist among the data. Here, the entire database is described in terms of small number of relatively simple structures. The conceptual level of abstraction is used by database administrators, who must decide what information is to be kept in the database. One conceptual view represents entire database. It is defined by conceptual schema.
3. **External Level:** This is the highest level in the three level architecture and closest to the user. It is also known as the view level. It describes only a part of the entire database. Many users of the database system will not be concerned with all of the information. They may need only a part of the entire database. To simplify their interaction with the database, view level is defined. The system may provide many views for the same database. It is defined by physical/external schema.

- An example in the real-world is a mobile or smartphone. The user does not need to understand the details of how a phone calls are placed or how the Internet works. These details are abstracted away from the phone's interface in order to make the phone easier and more effective to use. The design of the mobile or smartphone clearly separates the interface from the implementation. The user can learn the functions of the phone without ever having to know anything about how the phone works internally.

Advantages of Three-tier Architecture:

1. The main objective of it is to provide data abstraction. Data abstraction means to hide certain details of how the data is **stored** and **maintained.**
2. Same data can be accessed by different users with different customized views.
3. The user is not concerned about the physical data storage details.
4. Physical storage structure can be changed without requiring changes in internal structure of the database as well as users view.
5. Conceptual structure of the database can be changed without affecting end users.

1.4 VIEWS OF DATA - SCHEMAS AND INSTANCES, DATA INDEPENDENCE

Data-Schema:

- A database **schema** is a visual and logical architecture of a database created on a database management system.

- It provides a graphical view of the entire database architecture and structure. It provides a means for logically grouping and displaying database objects such as tables, fields, functions and relations.
- Database system supports three database schemas:
  1. **Physical schema:** It is at the lowest level i.e. at Physical level.
  2. **Logical schema:** It is at the next or intermediate level i.e. at Logical level.
  3. **Sub-schema/ External /view level Schema:** It is at the highest level i.e. at the View level.
- Analogy to the concept of data types, variables and values in programming language:

Consider a structure in C.

```
struct customer
{
    char customer_name [50];
    char social_security [50];
    char customer_street [50];
    char customer_city [50];
} cust;
```

- Here, cust is a variable of type customer structure. A database schema corresponds to the programming language type definition. The value of the variable in programming languages corresponds to an instance of a database schema.
- For example: In the following diagram, we have a scheme that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view (design) of a database as shown in the diagram below:

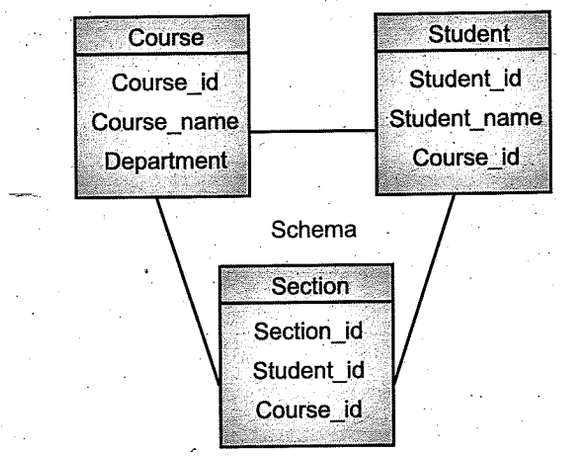


Fig. 1.3: Schema shows relationship between databases

- The design of a database at physical level is called **Physical schema.** This describes how the data stored in blocks of storage is described at this level.
- Design of database at logical level is called **Logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).
- Design of database at view level is called **View schema**. This describes end user interaction with database systems.

DBMS Instance:

- **Definition of instance**: Collection of information stored in the database at a particular (instance of time) moment is called an **instance** of the database.
- Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.
- For example, let us say we have a single table employee in the database. Today the table has 100 records, so today the instance of the database has 100 records. Let's say we are going to add another 200 records in this table by tomorrow so the instance of database tomorrow will have 300 records in table. In short, at a particular moment the data stored in database is called the instance that changes over time when we add or delete data from the database.

Data Independence:

- Data independence means the property to change the overall physical or logical structure of the data without changing the application program view of data i.e. the view created for users does not have any change.
- The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called **data independence**.
- There are two levels of data independence:
  1. Physical Data Independence.
  2. Logical Data Independence.

1. Physical Data Independence:

- It is the ability to modify the physical schema without causing application programs to be rewritten.
- In this physical layout and organization of data may be changed without changing either the overall logical structure of the data or the application program.
- It indicates that the physical storage structure used for storing data could be changed without causing a change in the conceptual view or any of the external views.

2. Logical Data Independence:

- It is the ability to modify the logical schema without causing application programs to be rewritten.
- It indicates that the conceptual scheme can be changed without affecting the existing external schemes.
- It is achieved by providing the external level or user view of the database.
- Logical data independence is more difficult to achieve than the physical data independence.

1.5 CENTRALIZED, CLIENT-SERVER SYSTEM, TRANSACTION SERVERS, DATA SERVERS, CLOUD BASED SERVERS

1.5.1 Centralized and Client/ Server system Architectures

- Historically, early database systems were based on a Centralized Model, in which the database resides on a single computer system that allows access by remote terminals. This is still the model used by many systems today.
- In this section, we will try to understand different aspects of Centralized and Client/Server architectures.

1.5.1.1 Centralized System

- Centralized database systems will run on a single computer system and do not interact with other computer systems. The centralized database's location is generally a server CPU or desktop or the mainframe computer which is accessed by the users through a computer network like LAN or WAN.
- *General-purpose computer system*: One to a few CPUs and a number of device controllers that are connected through a common bus that provide access to shared memory.
- *Single-user system* (e.g., personal computer or workstation): Desktop unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
- *Multi-user system*: More disks, more memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system via terminals. Multi-user systems are often called as server systems.
- 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.

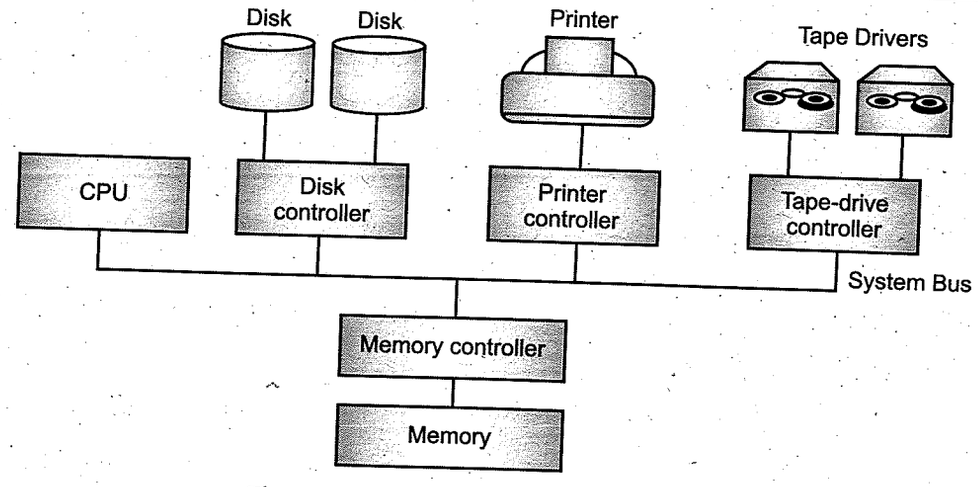


Fig. 1.4: A Centralized Computer System

Advantages of Centralized Database System:

1. The integrity of data is increased as the whole database is stored at a single physical location. In this way, it is easy to manage the data is most consistent and accurate.
2. The data redundancy is controlled in a centralized database. All the data is stored in one location and it cannot distribute on different places. So, it is easier to make sure all the data stored is not redundant. In this way redundancy easily controlled.
3. Centralized Database ensures a higher level of security. There are certain protocols established for the people associated or working with the data. So that the access of the data can be monitored and it reduces the amount of data leakage which in turn increases the data security.
4. As the data is available in one place, the single point of entry for the data is established which makes the implementation of the changes in data at once rather than implementing the changes multiple times.
5. The centralized database is inexpensive than other databases. It required less expense on maintenance.

Disadvantages of Centralized Database System:

1. A centralized system is totally vulnerable to failure of the centralized site. Thus, for example, a power failure at the site where the computer is located can shut down all access to the database, even at remote sites unaffected by the failure.
2. As the volume of transactions handled by the system grows, it can become increasingly difficult for a single system to handle the demand.
3. Another important issue that might generate from using the Centralised Database is data loss. As the data is present at one location, in case of any failure if the database is lost, and if there is no provision for back up, there is always the risk of losing the data. This might create an outage for a long time and the organization loses access to the data which will impact the business negatively.

1.5.1.2 Client/Server System

- The database systems functionality can be broadly divided into two parts: the front end and the back end.
- 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 *back end* manages access structures, query evaluation and optimization, concurrency control, and recovery.
- The interface between the front end and the back end is through SQL, or through an application program.

Components of Client-Server Architecture:

- There are three major components of Client/Server architecture :
  1. Server
  2. Client
  3. Network interface

1. **Server:** Server is DBMS itself. It consists of DBMS and supports all basic DBMS functions. Server components of DBMS are installed at server. It acts as monitor of all of its dients. It distributes work-load to other computers. Clients must obey their servers.

   Functions of Server: The server performs various functions, which are as follows.

   (i) It supports all basic DBMS functions.
   (ii) Monitor all its clients.
   (iii) Distribute work-load over clients.
   (iv) Solve problems which are not solved by clients.
   (v) Maintain security and privacy.
   (vi) Avoiding unauthorized access of data.

2. **Clients:** Client machine is a personal computer or workstation which provide services to both server and users. It must obey its server. Client components of DBMS are installed at client site. Clients are taking instructions from server and help them by taking their load. When any user want to execute a query on client, the client first take data from server then execute the query on his own hardware and returns the result to the server. As a result, server is free to do more complex applications.
3. **Network Interface:** Clients are connected to server by network interface. It is useful in connecting the server interface with user interface so that server can run his applications over his clients. In the client/server architecture, there are more than one server. Sometimes, a server is used Database Server, other as Application Server, other as Backup Server etc.

- In client/server computing, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server.

Types of Client/server architecture:

- Normally, there are three types of Client/Server architecture available in database management system. These are listed below.
  (a) Single-tier (1-tier) client/server computing model.
  (b) Two-tier client/server computing model.
  (c) Three-tier client/server computing model.

A. Single-tier Client/Server Architecture.

- Single tier architecture is the first type of Client/Server computing model. In single tire client/server computing model, the client server database system used on a personal computer. In single tire system, the database is centralized, which means the DBMS software and the data in one location and the dumb terminals were used to access the database management system.

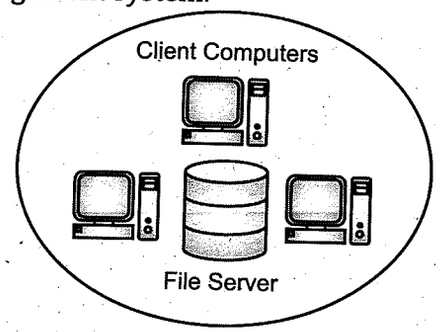


Fig. 1.5: 1-Tier Client/Server Architecture

Advantage of Single-Tier:

1. The data is easily and quickly available since it is located in the same machine.

Disadvantage of Single-Tier:

1. This architecture is completely not scalable. Only one user can access the system at a given time through the local client.

B. Two-tier Client/Server Architecture:

- The two-tier architecture primarily has two parts, a Client tier and a Server tier. The client tier sends a request to the server tier and the server tier responds with the desired information.
- An example of a two-tier client/server structure is a web server. It returns the required web pages to the clients that requested them.

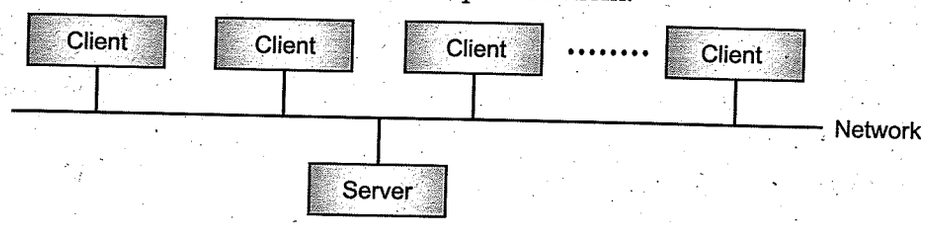


Fig. 1.6: Two - Tier Client/Server Architecture

Advantages of Two - Tier Client/Server Structure:

1. This structure is quite easy to maintain and modify.
2. The communication between the client and server in the form of request response messages is quite fast.

Disadvantage of Two - Tier Client/Server Architecture:

1. If the client nodes are increased beyond capacity in the architecture, then the server is not able to handle the request overflow and performance of the system degrades.

C. Three - Tier Client/Server Architecture:

- The three-tier architecture has three layers namely Client, Application and Data layer.
  - The client layer is the one that requests the information. In this case it could be the GUI, web interface etc.
  - The application layer acts as an interface between the client and data layer. It helps in communication and also provides security.
  - The data layer is the one that actually contains the required data.

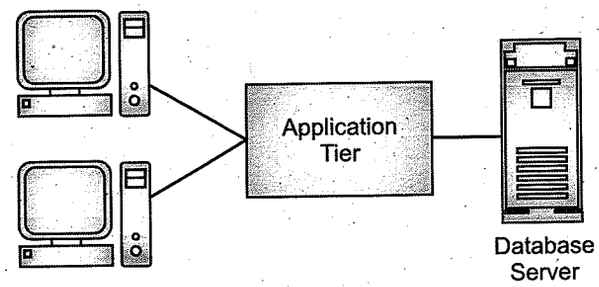


Fig. 1.7: Three - Tier Client/Server Architecture

Advantages of Three - Tier Client/Server architecture:

1. The three-tier structure provides much better service and fast performance.
2. The structure can be scaled according to requirements without any problem.
3. Data security is much improved in the three-tier structure.

Disadvantage of Three - Tier Client/Server architecture:

1. Three - tier Client/Server structure is quite complex due to advanced features.

1.5.2 Server System Architectures

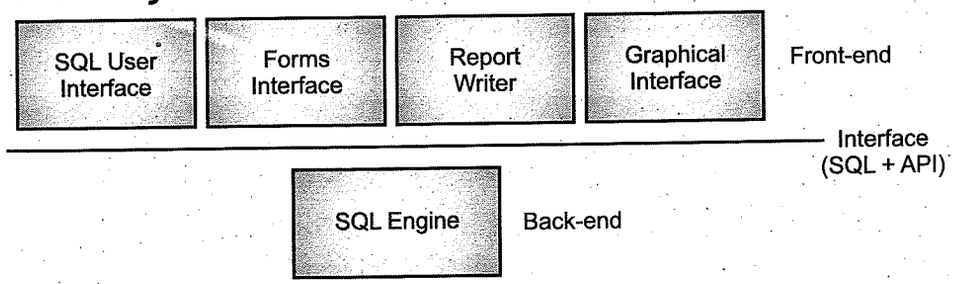


Fig. 1.8: Front-end and Back-end functionality

- Server systems can be broadly categorized into two types:
  1. **Transaction servers:** These servers are widely used in relational database systems.
  2. **Data servers:** These servers used in object-oriented database systems.

1.5.2.1 Transaction System

- A transaction server is a specialized type of server that manages the operations of software based transactions or transaction processing. It manages application and database transactions on a network or Internet, within a distributed computing environment.
- A transaction server may also be referred to as a Transaction Processing System (TPS).
- This is also called **Query server systems** or **SQL server systems**.
  - Clients send requests to the server.
  - Transactions are executed at the server.
  - Results are shipped back to the client.
- Requests are specified in SQL and communicated to the server through a Remote Procedure Call (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- Open Database Connectivity (ODBC) is a C language application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.
- JDBC standard is similar to ODBC, for Java.

Transaction Server Process Structure:

- A typical transaction server consists of multiple processes accessing data in shared memory.

1. **Server process:**
   - These receive user queries (transactions), execute them and send results back.
   - Processes may be multithreaded, allowing a single process to execute several user queries concurrently.
   - Typically multiple multithreaded server processes.
2. **Lock manager process** includes lock grant, lock release, and deadlock detection.
3. **Database writer processes:** There are one or more processes that output modified buffer blocks back to disk on a continuous basis.
4. **Log writer process:**
   - Server processes simply add log records to log record buffer.
   - Log writer process outputs log records to stable storage.
5. **Checkpoint process** performs periodic checkpoints.

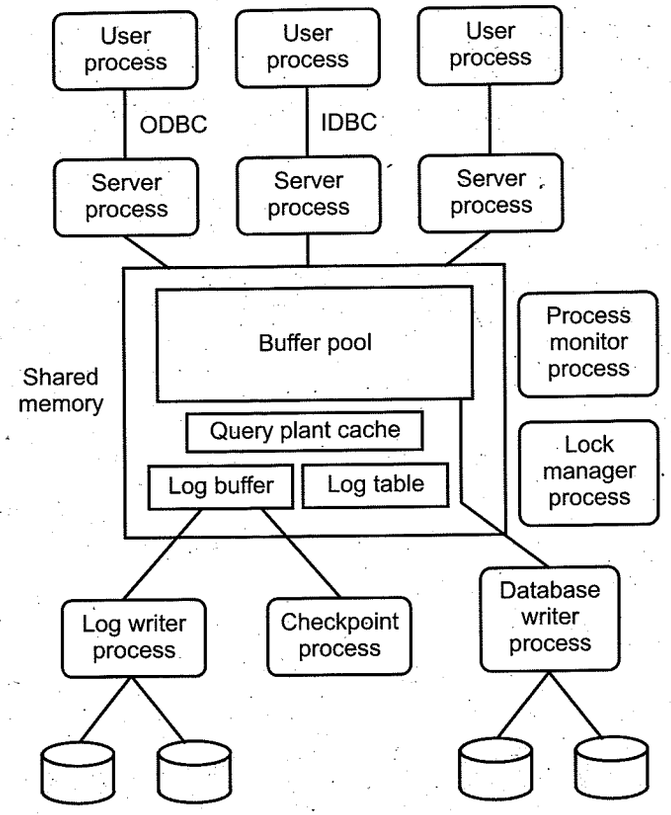


Fig. 1.9: Shared Memory and Process Structure

6. **Process monitor process:**
   - Monitors other processes, and takes recovery actions if any of the other processes fail.
   - For example, aborting any transactions being executed by a server process and restarting it.

- The shared memory contains all shared data, such as:
  - Buffer pool
  - Lock table
  - Log buffer containing log records waiting to be output to the log on stable storage.
- All database processes can access shared memory. To ensure that no two processes are accessing the same data structure at the same time, databases systems implement mutual exclusion using either: Operating system semaphores or Atomic instructions such as test-and-set.
- To avoid overhead of inter-process communication for lock request/grant, each database process operates directly on the lock table instead of sending requests to lock manager process.
- Lock manager process still used for deadlock detection.

1.5.2.2 Data Servers

- A data server defines the physical connection to a database. The primary function of a data server is to store data and provide access to the other systems.
- A data server is also known as a database server.
- The data server connection specifies the parameters needed to the database such as the location of the database and the timeout duration. It can include authentication information.
- Data servers are used in high-speed LANs, mostly in cases where:
  - The clients are comparable in processing power to the server.
  - The tasks to be executed are computed intensive.
  - Data are shipped to clients where processing is performed, and then shipped results back to the server.
  - This architecture requires full back-end functionality at the clients.
  - These are used in many object-oriented database systems.
- **Issues** arise in such architecture are:

(a) Page-shipping versus item-shipping:

- If the unit of communication is a single item, the overhead of message passing is high compared to the amount of data transmitted.
- Worth prefetching related items along with requested item.
- Page shipping can be thought of as a form of prefetching.

(b) Locking:

- Overhead of requesting and getting locks from server is high due to message delays.
- Can grant locks on requested and prefetched items; with page shipping, transaction is granted lock on whole page.

(c) Data Caching:

- Data can be cached at client even in between transactions.
- But check that data is up-to-date before it is used (cache coherency).
- Check can be done when requesting lock on data item.

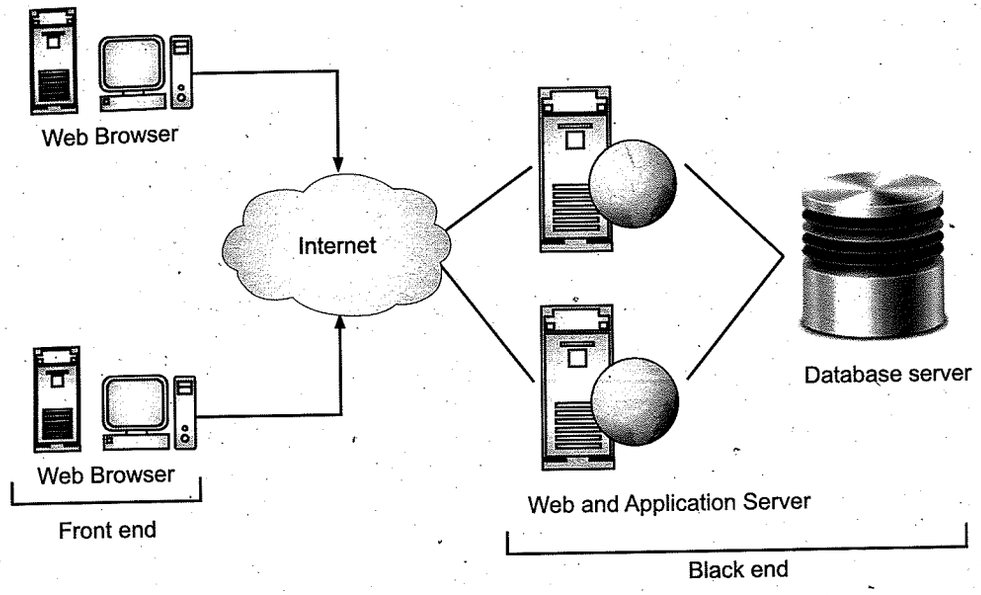
(d) Lock Caching:

- Locks can be retained by client system even in between transactions.
- Transactions can acquire cached locks locally, without contacting server.
- Server calls back locks from clients when it receives conflicting lock request. Client returns lock once no local transaction is using it.

1.5.3 Cloud-based Servers

- Servers are usually owned by the enterprise providing the service. But there is an increasing trend for Service providers to rely at least in part upon servers that are owned by a "third party" that is neither the client nor the service provider.

- One model for using third-party servers is to outsource the entire service to another company that hosts the service on its own computers using its own software. This allows the service provider to ignore most details of technology and focus on the marketing of the service.



1.10: Cloud-based Server Architecture

- Another model for using third-party servers is **Cloud computing**, in which the service provider runs its own software, but runs it on computers provided by another company. Under this model, the third party does not provide any of the application software; it provides only a collection of machines. These machines are not "real" machines, but rather simulated by software that allows a single real computer to simulate several independent computers. Such simulated machines are called virtual machines.
- A cloud server is a virtual server (rather than a physical server) running in a cloud computing environment. It is built, hosted and delivered via a cloud computing platform via the internet, and can be accessed remotely. They are also known as virtual servers.

Advantages of a cloud server:

1. A cloud server gives the business user stability and security because any software problems are isolated from the environment. Other cloud servers won't impact on cloud server and vice versa. If another user overloads their cloud server, this will have no impact on user's cloud server, unlike with physical servers.
2. Cloud servers are stable, fast and secure. They avoid the hardware issues seen with physical servers, and they are likely to be the most stable option for businesses wanting to keep their IT budget less.

3. Cloud servers provide a faster service. User will get more resources and a faster service than he would for a similar price of physical server. A cloud-hosted website will run faster.
4. User gets scalability with cloud servers. It is very easy and quick to upgrade by adding memory and disk space, as well as being more affordable.

Summary

- A database is a collection of related data which represents some aspect of the real world.
- Database Management System (also known as DBMS) is software for storing and retrieving users' data by considering appropriate security measures.
- The ANSI-SPARC Architecture, where ANSI-SPARC stands for American National Standards Institute, Standards Planning And Requirements Committee, is an abstract design standard for a Database Management System (DBMS).
- A database **schema** is a visual and logical architecture of a database created on a database management system.
- Collection of information stored in the database at a particular (instance of time) moment is called an **instance** of the database.
- The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called **data independence**.
- Servers can be either transaction servers or data servers, although the use of transaction servers greatly exceeds the use of data servers for providing database services.
- In client/server computing, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server.
- Centralized database systems will run on a single computer system and do not interact with other computer systems.
- Transaction servers have multiple processes, possibly running on multiple processors. So that these processes have access to common data, such as the database buffer, systems store such data in shared memory.
- Data-server systems supply raw data to clients. Such systems strive to minimize communication between clients and servers by caching data and locks at the clients.
- A Cloud server is a hosted, and typically virtual, compute server that is accessed by users over a network. Cloud servers are intended to provide the same functions, support the same Operating Systems(OS) and applications, and offer performance characteristics similar to traditional physical servers that run in a local data centre.

Check Your Understanding

1. DBMS is a collection of _____ that enables user to create and maintain a database.
   (a) Keys (b) Translators
   (c) Program (d) Language Activity
2. The collection of information stored in a database at a particular moment is called as _____.
   (a) schema (b) instance of the database
   (c) data domain (d) independence
3. The view of total database content is _____.
   (a) Conceptual view (b) Internal view
   (c) External view (d) Physical view
4. Which database level is closest to the users?
   (a) External (b) Internal
   (c) Physical (d) Conceptual
5. The ability to change the conceptual schema without affecting the external schemas or application programs is known as _____.
   (a) Physical data independence (b) Logical data independence
   (c) Program independence (d) Data abstraction
6. Which of these is not an advantage of database systems?
   (a) Centralized data management (b) Program data independence
   (c) Data redundancy (d) Data abstraction
7. Which one of the following is not a DBMS tool?
   (a) SQLite (b) PHP
   (c) MongoDB (d) Oracle
8. _____are known as virtual servers.
   (a) Data servers (b) Transaction servers
   (c) Cloud servers (d) None of the above
9. In which of the databases, both the database and most of its DBMS reside remotely?
   (a) in-memory database (b) active database
   (c) cloud database (d) all of the above
10. A database server is responsible for which of the following?
    (a) Database storage (b) Data processing logic
    (c) Data presentation logic (d) All of the above

ANSWER KEY

1. (c)	2. (d)	3. (a)	4. (a)	5. (b)
6. (d)	7. (b)	8. (c)	9. (c)	10. (a)

Practice Questions

Q.I: Answers the following questions in short.

1. Enlist various applications of DBMS.
2. Define data schema and instances.
3. What is data independence? What are its types?
4. What is Client/Server database system?
5. List types of server systems.

Q.II: Answers the following questions.

1. What is DBMS? Describe the need of DBMS in detail.
2. With the help of diagram describe overall structure of DBMS.
3. Explain Client/Server System architecture.
4. Explain two-tier Client/Server structure. Also state the advantage and disadvantage of it.
5. Explain three-tier Client/Server structure. Also state the advantage and disadvantage of it.
6. Explain Transaction server's process structure with diagram.

Q.III: Write a short note on:

1. Centralized systems
2. Three-tier client/server architecture of DBMS
3. Characteristics of DBMS
4. Cloud based servers
5. Data servers

❖❖❖

2... Data Modelling and Relational Database Design

Objectives...

- To learn about Data Modelling using E-R Diagram.
- To study Relational data model.
- To know about Database Design using E-R and E-R to Relational.
- To get information of Codd's rules and Normalization.

2.1 INTRODUCTION OF DATA MODELLING USING E-R DIAGRAM

Data Model:

- A data model is a conceptual tool that can be used to describe the structure of a database.
- While designing a database, a developer has to make decisions about the data to be stored in a database. Also the structure and relationship of that data has to be decided. So developer uses some methods to create a database for the user and for the system implementation. These methods are called as model.
- A **model** defines the method of storing and retrieving data. It is an abstraction process that hides the details which are not required while highlighting details required to the applications at hand.
- **A data model consists of:**
  - A named logical unit i.e. record type and data item.
  - Relationship among these logical units.
  - Data item is logical unit of data and record type is collection of data items.

- We can say, a data model is description of a container in which data and its methods of storing and retrieving data are available. Actually, it provides abstraction of database application.
- Data models are classified into three categories.
  1. Object Based Logical Data Model: This model is used to describe data at the logical and view level of a database. It is used to describe logical data structure.
  2. Record Based Logical Data Model: This model is also use to describe data at logical and view level. It is used to describe the logical database structure in terms of records.
  3. Physical Data Model: It describes about the way of data stored in secondary storage device by representing information such as record formats, record ordering, access path etc.

E-R diagram:

- E-R diagram was developed in 1970 by **Dr. Peter Chen** and others. They simplified the representation of large and complex data storage concepts using the E-R diagrams.
- An E-R model is based on a perception of a real world which consists of a collection of basic objects called entities and relationship among these objects.
- An E-R model is normally expressed as an E-R diagram which is a graphical representation of a particular database.
- Elements of the E-R model:
  1. Entity.
  2. Entity sets.
  3. Attributes.
  4. Relationships and relationship sets.

2.1.1 Representation of Entities

- An **entity** is an **'object'** in the real world which has its own properties and it is distinguishable from other objects.
- An entity can be concrete or abstract.
- An entity is represented by rectangle shape [ ].

For example:

[Employee] → Here, entity is employee

- An entity is represented by a rectangle in E-R diagram. An entity has set of properties called **attributes** which holds value in it. An entity must have some attributes through which it is uniquely identified.

For example: [Emp_code] will identify an employee code.

- An **entity set** is a set of entities of the same type that share the same **properties** or attributes. For example: The set of all students is defined as entity set student.
- **Entity** sets need not be disjoint. For example: A person entity could be in both the customer and employee sets.

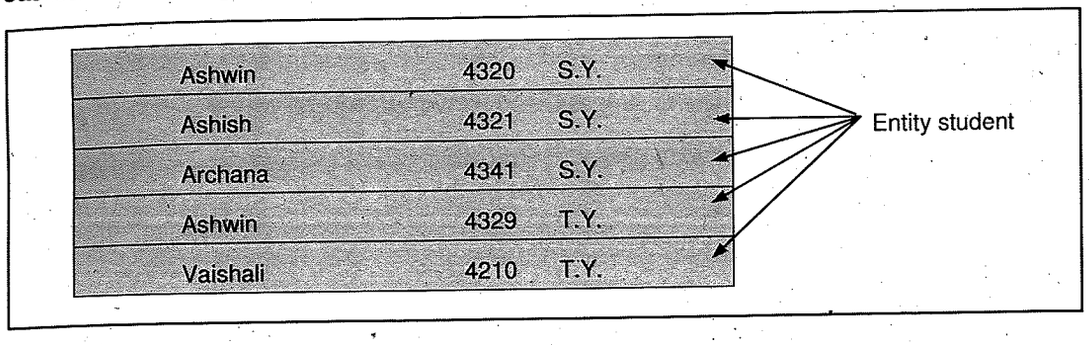


Fig. 2.1: Representation of Student Entity Set

2.1.2 Attributes

- An attribute of an entity is a particular property that describes the entity.
- An attribute is represented by ellipse shape ( ).

For example:

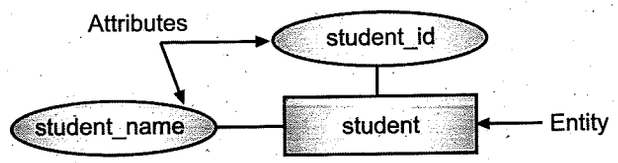


Fig. 2.2: Attributes in the set

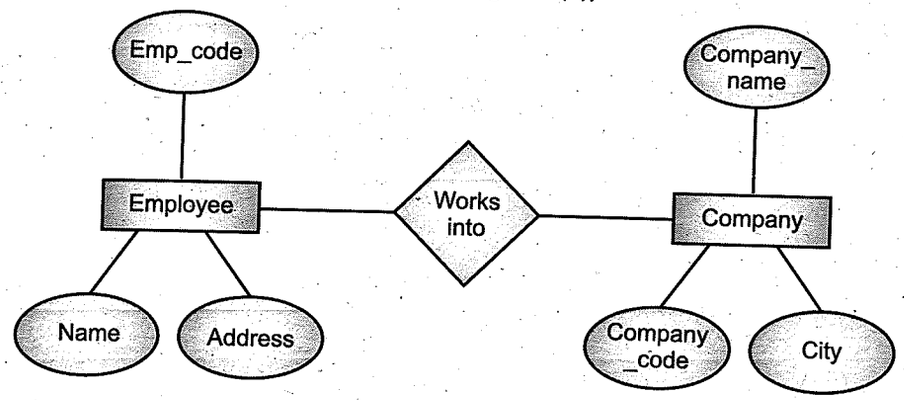
- Once the entity is identified, it is described in real terms or through its attributes.
- An attribute is description that is used to identify, qualify, classify, or otherwise express the set of an entity occurrence or a relationship.
- For each attribute, set of permitted values are available called as domain. The attributes which uniquely defines the occurrence of an entity are called **primary keys**. If no attribute works as a primary key then a new attribute is defined for that purpose. For example,

Table 2.1: 'Disease' database

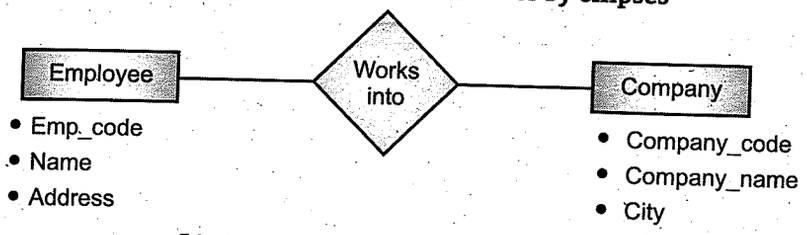
Disease name	Causing organism
Typhoid	Salmonella Typhi
Dysentery	Shigella dysentriae

- No primary key is available. Then we can consider **disease-no** as extra field or attribute.

- An attribute must appear in only one place in the model. Either it may be required or optional. When the attribute is required, it holds a value. When it is optional, it may or may not have a value for it.
- For example, 'plant' Entity. Its attributes are plant_name, plant_type, date_of_ acquisition and pot_size. Other fields are required. But pot size is optional because some plants do not come in pots.
- In E-R diagram, an attribute is represented in two different ways.
  1. By ellipses attached to entities, (Refer Fig. 2.3 (a)).
  2. By listing the attribute as text, (Refer Fig. 2.3 (b)).



(a): Representation of attributes by ellipses



(b): Representation of attribute by text

Fig. 2.3

Types:

- An attribute may be of following types:
  1. **Simple attribute:** It represents a single property i.e. these are not divided into subparts. These are also called as atomic attributes.

     For example: 

  2. **Composite attributes:** The attributes can be divided into subparts. This helps us to group together the related attributes. This may appears as a hierarchy.

     For example, Address is the composite attribute having street address, city, state, zip as simple attributes as shown in Fig. 2.4.

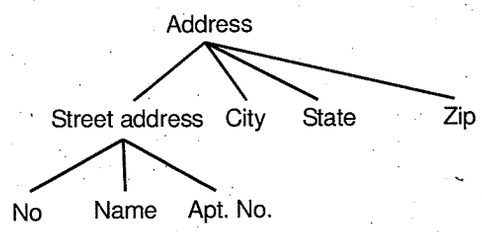


Fig. 2.4: Composite Attribute

3. **Single valued attributes:** The attributes which contains one value in their column is called single valued attributes. For example: roll_no, name, amount, age has only one value.
4. **Multi-valued attributes:** The attributes which contains more than one value for a specific entity is called multi-valued attributes.

   For example:

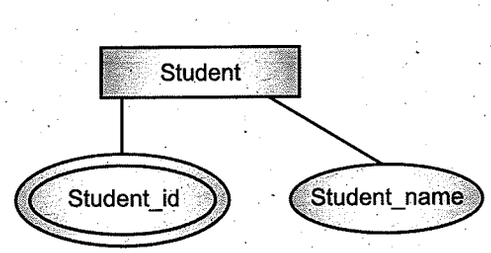
   (i) Person has many phone numbers.

   (ii) Number of dependents on an employee.

   The multivalued attributes are represented by the following symbol.

(a) Symbol of Multivalued Attribute

For example:



(b) Multivalued Attributes

Fig. 2.5

5. **Null attributes:** The attribute without any value is called as **null attribute**. Null value is used when entity does not have a value for an attribute. There are two assumptions done for this null attribute.

   Sometimes, when there is no value we say "**not applicable**" or sometimes we may say "**value is missing**".

   Let's see the example. Suppose, employee have no dependents and the name of dependent is null, it means, this is not applicable.

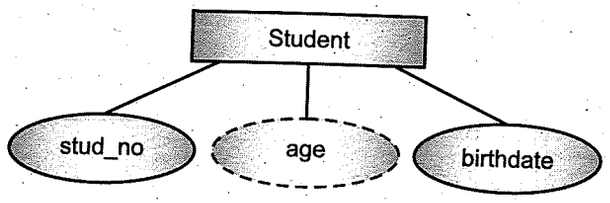
   If say employee_code or social security number is null, it means, it is missing.

   The attribute *phone_no* of employee can have value NULL, if employee does not have telephone connection.

6. **Derived attributes:** The value of this attribute can be derived from values of other related attributes or entities. Finding marks or grade or percentage of a student is the best example of derived attribute. Given an attribute *birthdate*, *age* can be derived from this attribute for an employee. So *age* is derived attribute and *birthdate* is called stored attribute. It is represented as dotted ellipse or ovals as shown in Fig. 2.6.

(a) Symbol of Derived Attribute

For example:



(b) Derived Attribute

Fig. 2.6

- The concept of an entity set corresponds to the programming language notion of type definition.
- A variable of a given type has a particular value at a given instant in time. Thus, a variable in programming language corresponds to the concept of an entity in the E-R model.
- A database thus includes a collection of entity sets each of which contains any number of entities of the same type. For example, bank database that consists of two entities sets: *customer* and *account*. In this section, we shall be dealing with five entity sets with unique attribute names.
  - **branch**, the set of all branches of a bank. Each branch is described by attributes branch_name, branch_city and assets.
  - **customer**, the set of all people who have account at bank. Its attributes are customer_name, social_security, street and customer_city.
  - **employee**, the set of people works in bank. Each employee has attributes employee_name and phone_number.
  - **account**, the set of all accounts maintained in the bank and attributes are account_number and balance.
  - **transaction**, the set of all account transactions executed in the bank. Attributes are trans_no, data and amount.

2.1.3 Relationship and Their Type

Relationship:

- A relationship is association or linkage among several entities.
- We can define relationship as "An association between several entities".

For example: The relationship between Employee and Dept is shown in the Fig. 2.7(a). The relationship between customer and sales order is shown in the Fig. 2.7(b). Similarly the relationship between customer and account is shown in the Fig. 2.7(c).

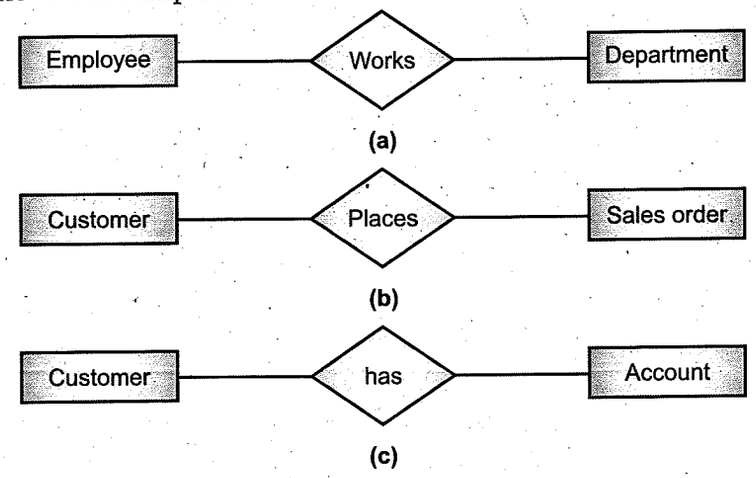


Fig. 2.7: Relationship

- The relationships are represented by diamond shapes  (See, Fig. 2.7).

Relationship Set:

- A relationship set is a set of relationships of the same type. Formally speaking, it is a mathematical relation on $n \geq 2$ possibly non-distinct sets.
- If E1, E2, E3 ..., En are entity sets, then a relationship set S is a subset of,

  $$\{(e1, e2, ..., en) \mid e1 \in E1, e2 \in E2, ..., en \in En\}$$

  Where, {e1, e2, ..., en} in a relationship.

For example: Consider the two entity sets Doctor and Patient.

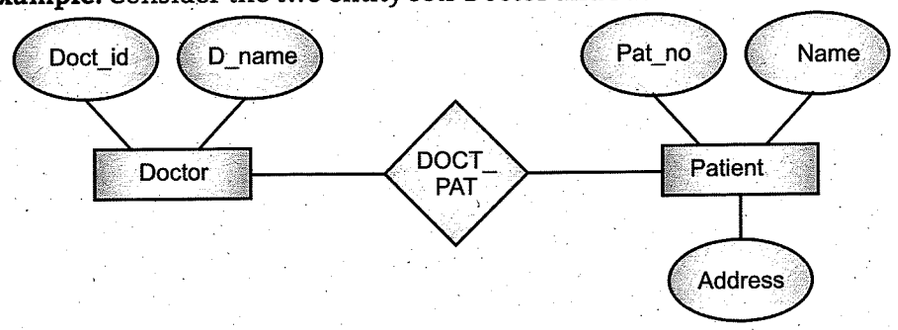


Fig. 2.8: Entity Sets 'Doctor and Patient'

- We can define the relationship set DOCT_PAT to denote the association between above named entities. This represents real-world entities.

DOCT_ID	D_NAME
01	Sharma
02	Sakhare
03	Kale

PAT_NO	NAME	ADDRESS
101	Bina	Pune
102	Tina	Vadgaon
103	Rina	Warje

Fig. 2.9: Relationship Set DOCT_PAT

- From Fig. 2.9, we can say doctor Sharma handles 2 patients i.e. Bina and Tina.

Types of Relationship:

- There are different types of relationship occurs amongst the entities, depending upon the cardinality. **Cardinality** express or shows number of entities in the entity set.

  For example: If entity set is customer – loan – Bank

  ∴ Cardinality is 3.
- Even though generally we consider binary relationships only i.e. customer – loan, loan - Bank etc.
- There are four types of relationships between data.
  1. One-to- One relationship
  2. One-to- Many relationship
  3. Many-to- One relationship
  4. Many-to- Many relationship

2.1.4 Mapping Cardinality

- Mapping cardinality or cardinality ratio expresses the number of entities to which another entity can be associated via a relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets.
- For a binary relationship set R between two entity sets A and B, the mapping cardinality must be one of the following.

1. **One-to-One (1:1):** An entity in A is associated at the most one entity in B and then entity in B is also associated with at the most one entity in A. [Refer Fig. 2.10].

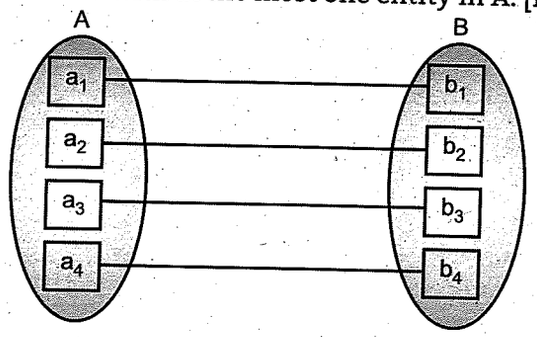


Fig. 2.10: One-to-One Relationship

For example, one class has one Roster [Refer Fig. 2.11].

Fig. 2.11: Representation of One-to-One Relationship

2. **One-to-Many (1:M):** An entity in A is associated with any number of entities in B and entity in B is associated with at the most one entity in A. The 1: M relationship is represented using 'parent-child' concept. Here 'one' side of relationship is called *parent* and 'many' side of relationship is called as *child*, (See Fig. 2.12).

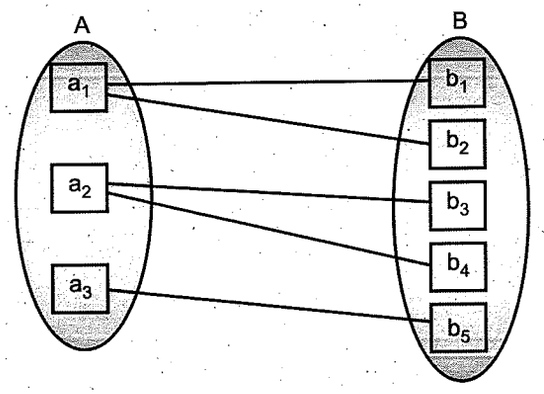


Fig. 2.12: One-to-Many Relationship

For example: One student offers many subjects, [See Fig. 2.13 (a)]. Similarly, one manager manages many employees, [See Fig. 2.13 (b)].

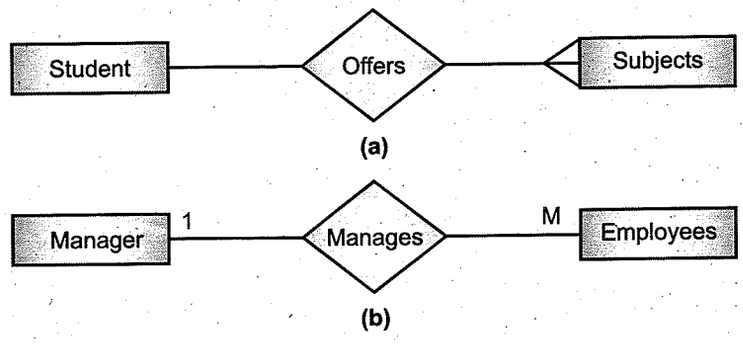


Fig. 2.13: Representation of One-to-Many Relationship

3. **Many-to-One (M:1):** An entity in A is associated with at most one entity in B and an entity in B is associated with any number of entities in A, (Refer Fig. 2.14).

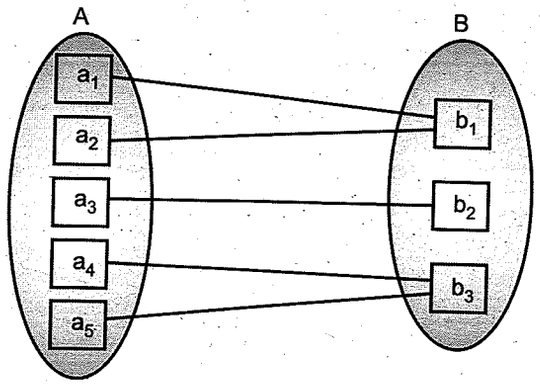


Fig. 2.14: Many-to-One Relationship

For example: Many politicians are present in a party, [See Fig. 2.15 (a)]. Many employees are working in a department [Refer Fig. 2.15 (b)]. Means one department can have many employees and one employee in one department.

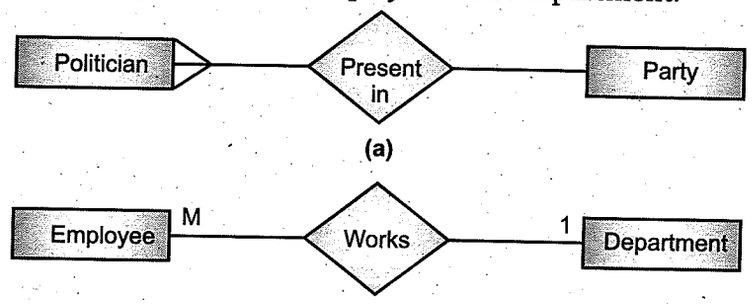


Fig. 2.15: Many-to-One Relationship

4. Many-to-Many (M:M): Entities in A can be associated with any number of entities in B and vice versa also, [Refer Fig. 2.16].

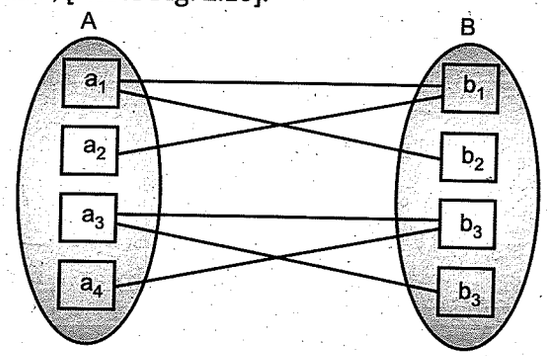


Fig. 2.16: Many-to-Many Relationship

For example: Many teachers' conducts many tests [see Fig. 2.17 (a)]. Similarly many doctors work in many Hospitals, [See Fig. 2.17 (b)]. Means one doctor in many hospitals and one hospital has many doctors. Therefore, relationship is many-to-many.

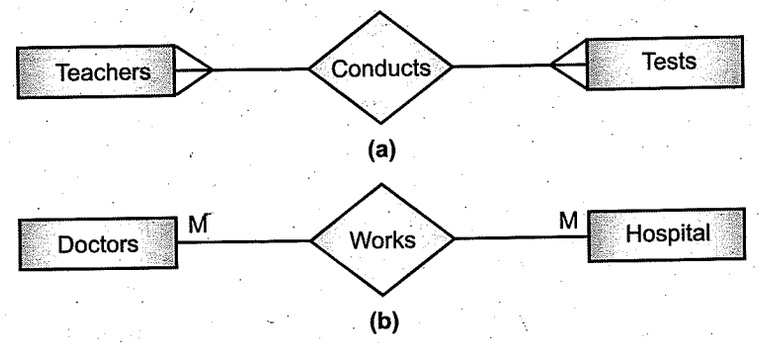


Fig. 2.17: Many-to-Many relationship

- **Representation of Mapping Cardinalities:** Mapping cardinality is indicated by directed line (→) or undirected line (—).
  - Consider following two entities: Customer – { cust_name, social_security, address} and Account – { acc_no, balance}
  - Relationship set: Deposit with attributes {social_security, acc_no}.

1. If the relationship is **Many-to-Many** i.e. one customer may have any number of accounts and one account can be shared by any number of customers, then this cardinality is represented as follows:

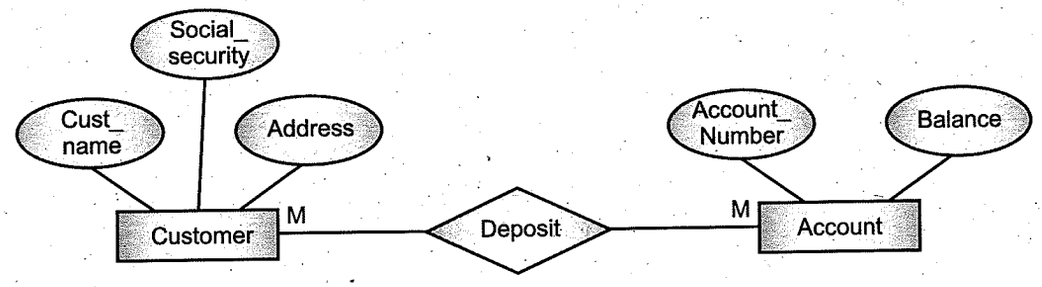


Fig. 2.18 (a): Many-to-Many Relationship

2. If the relationship is **One-to-Many** i.e. one customer may have any number of accounts, but one account belongs to only one customer then this cardinality is represented as follows:

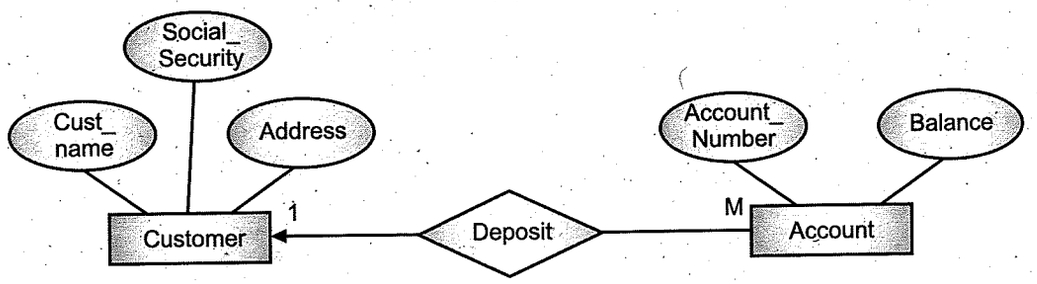


Fig. 2.18 (b): One-to-Many Relationship

3. If the relationship is **Many-to-One** i.e. many customers can share one account, then this cardinality is represented as follows:

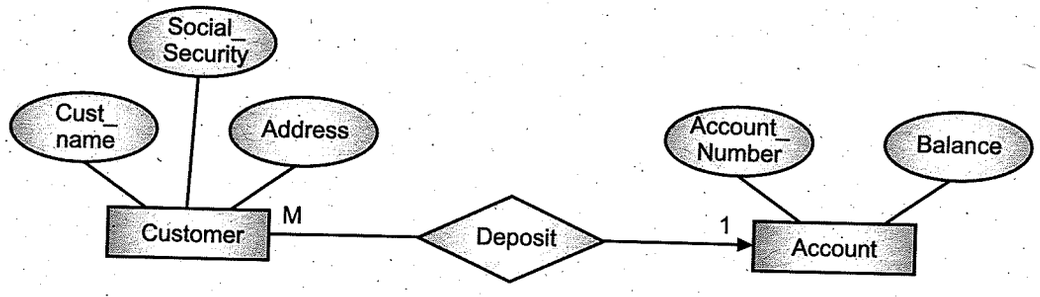


Fig. 2.18 (c): Many-to-one Relationship

4. If the relationship is **One-to-One** i.e. each customer has one account and each account belongs to one customer, then this cardinality is represented as follows:

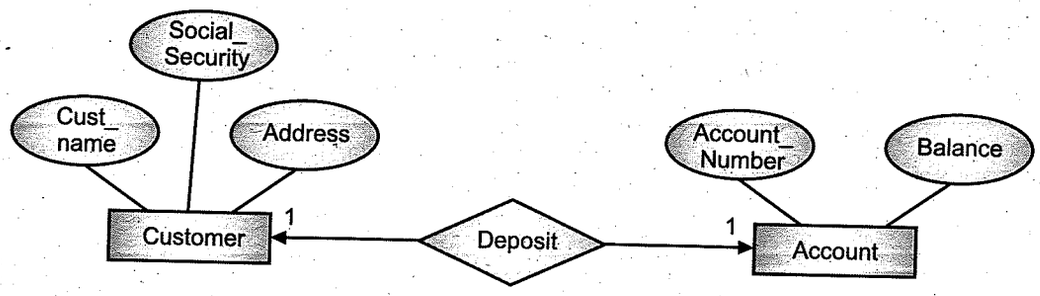


Fig. 2.18 (d): One-to-One Relationship

- All above are the types of relationship depending upon cardinality.
- There are three more types of relationship available.
  1. Binary relationship.
  2. Unary relationship.
  3. Ternary relationship.

1. **Binary relationship:** It has involvement of two entities only. Therefore, the degree of relationship is 2. One entity may be related with 1 : 1, 1 : M, M : M cardinalities with another entity.

   For example:

   Many products can be ordered by many customers. Here, product and customer are two entities, [Refer See Fig. 2.19 (a)]. Similarly, one customer can have many accounts so again two entities involved, [Refer Fig. 2.19 (b)] and therefore, referred as binary relationship.

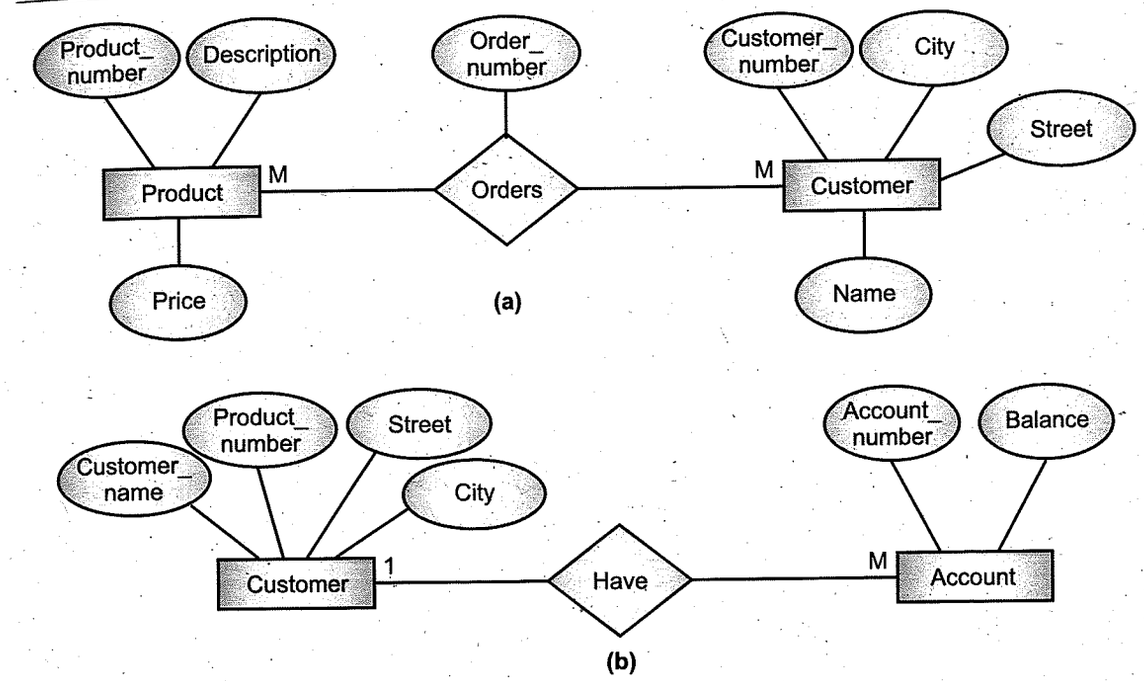


Fig. 2.19: Binary relationship

- In E-R diagram, primary key is represented by the symbol shown in Fig. 2.20.

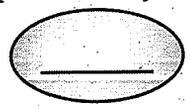


Fig. 2.20: Primary key representation

2. **Unary relationships:** This represents the association between the occurrences of **single** entity. Therefore, the degree is 1, [Refer Fig. 2.21].

   For example:

   (a) Unary Person 1: 1 person (Marriage).

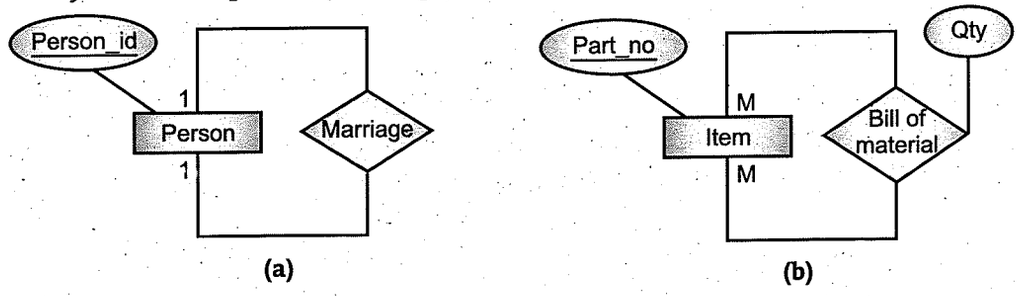


Fig. 2.21: Unary Relationship

   (b) Unary item M: M item (Bill of material)

3. **Ternary Relationship:** It refers to the occurrence of three entities at the same time and degree is six.

For example:

(i) An employee works into a branch and his job is say manager. If we write it as binary relationship, then Employee-Branch and Employee-Job will create some problems for validity, [See Fig. 2.22 (a)].

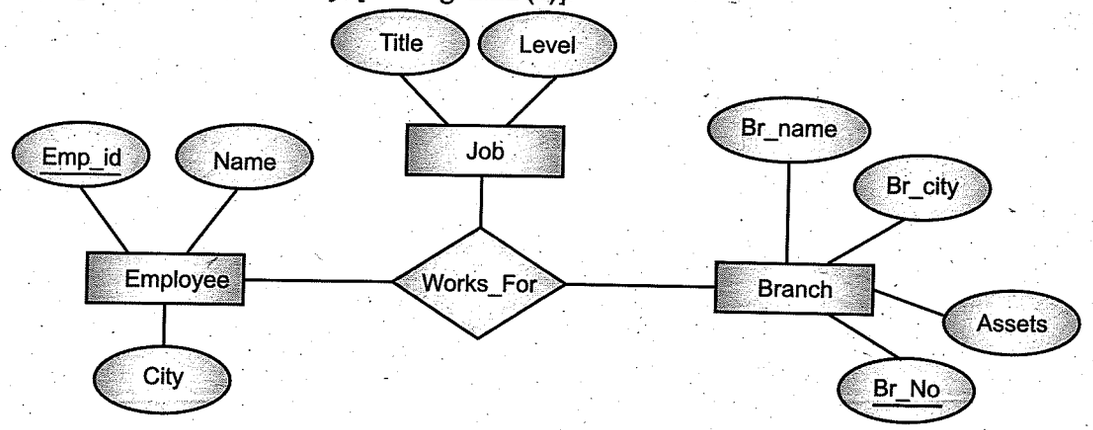


Fig. 2.22 (a)

(ii) Ternary relationship is available with Customer, Branch and Account as shown in the Fig. 2.22 (b).

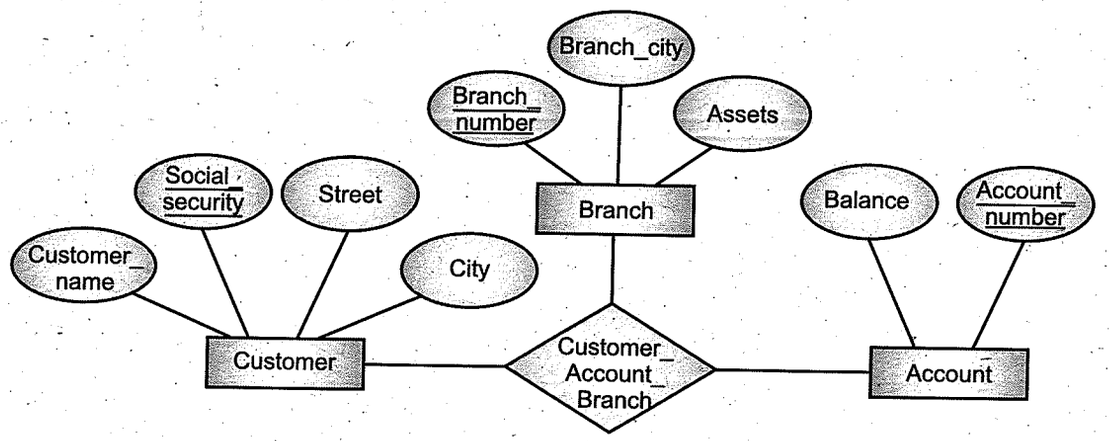


Fig. 2.22 (b): Ternary Relationship

Weak entity and Strong Entity Sets:

- Sometimes, there is no attribute in an entity which shows uniqueness of entity. An entity set may not have sufficient attributes to form a primary key. Entity set is termed as **weak entity set**.
- The entity set which has a primary key is known as **strong entity set**.
- A weak entity set is indicated by double lined rectangle.

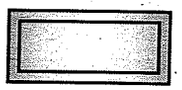


Fig. 2.23: Symbol of Weak Entity Set

- **Example:** In a bank, customer has account. Therefore, Customer and Account are related with each other [Refer Fig. 2.19 (b)]. Customer does some transactions in a bank on the account. In transaction entity, transaction_no is considered as primary key. For a customer deposit and withdrawal is done and key value starts from one because it is typically in a sequential number.

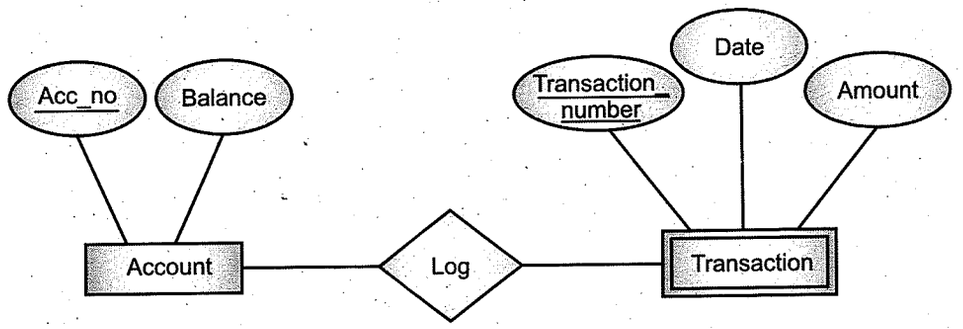


Fig. 2.24: Account keeps Log of transaction

- For each customer, the Transaction_number starts at 1. See following table, [Refer Fig. 2.25].

Acc_no	Balance
111	10000
222	5000
333	8000

Transaction_Number	Date	Amount
1	28/3/2003	
2	30/3/2003	
1	3/4/2003	

One customer

2nd customer

Fig. 2.25: For each customer, the Transaction_number starts at 1

- Eventhough transactions are distinct, same transaction_number is used for deposit and withdrawal for different customers. Thus, transaction_number is not a unique value. So it is referred as weak-entity. Therefore, the relationship is also weak.

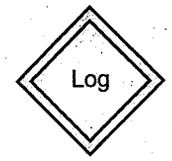


Fig. 2.26: Weak Relationship

- To make a weak-entity set meaningful it must be associated with another entity set called as **owner entity set**. If we consider Acc_no with Transaction_number, it will show a healthy relationship between account and transaction, [Refer Fig. 2.27].

Acc_no	Transaction_number	Date	Amount
111	1	28/3/2003	+500
222	1	3/4/2003	−1000

Fig. 2.27: Strong Relationship through Weak Entity

2.1.5 Generalization

- Generalization is abstracting process of viewing set of objects as a single general class. It concentrates on the general characteristics of consequent set while suppressing or ignoring their differences.
- It means generalization is the result of taking the union of two or more entity sets to produce a higher level entity set, [Refer Fig. 2.28].

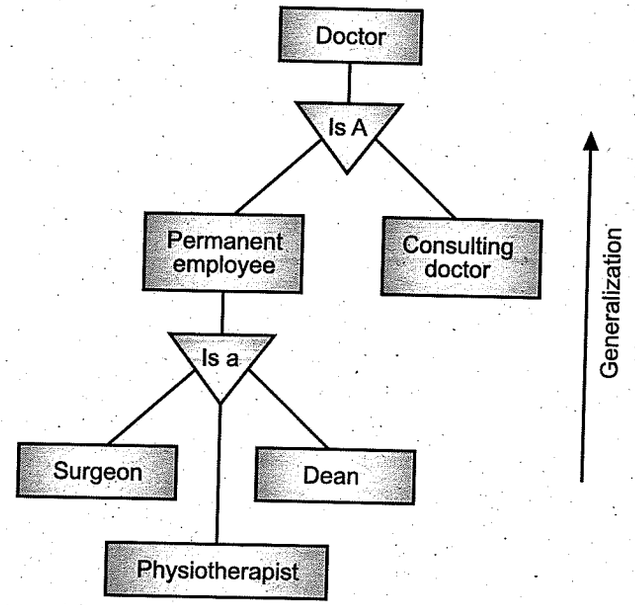


Fig. 2.28: Generalization

- This line implies that a doctor has to be either a Permanent employee or a Consulting doctor, nothing else. The entities are limited in generalization.
- It is a containment relationship that exits between a higher level entity set and one-or-more lower level entity sets. Attributes of higher level entity set and of lower level entity set are common. This commonality is expressed by generalization.

2.1.6 Specialization

- Specialization, as the name suggests, is a process of specializing an entity type into a more specified entity. In this process, we specialize a higher-level entity type by adding some additional attributes to the entity type. In this approach, we add some additional attributes and specialize a higher-level entity type into some other entity type.
- It is a top-down approach in which a higher-level entity is broken into smaller entities.
- **Example:** If we have a person entity type who has attributes such as Name, Phone_no, Address. Now, suppose we are making software for a shop then the person can be of two types. This Person entity can further be divided into two entities i.e. Employee and Customer. How we will specialize it? We will specialize the Person entity type to Employee entity type by adding attributes like Employee_id and Salary. Also, we can specialize the Person entity type to Customer entity type by adding attributes like Customer_id, Email, and Credit. These lower-level attributes will inherit all the properties of the higher-level attribute. The Customer entity type will also have attributes like Name, Phone, and Address which was present in the higher-level entity.

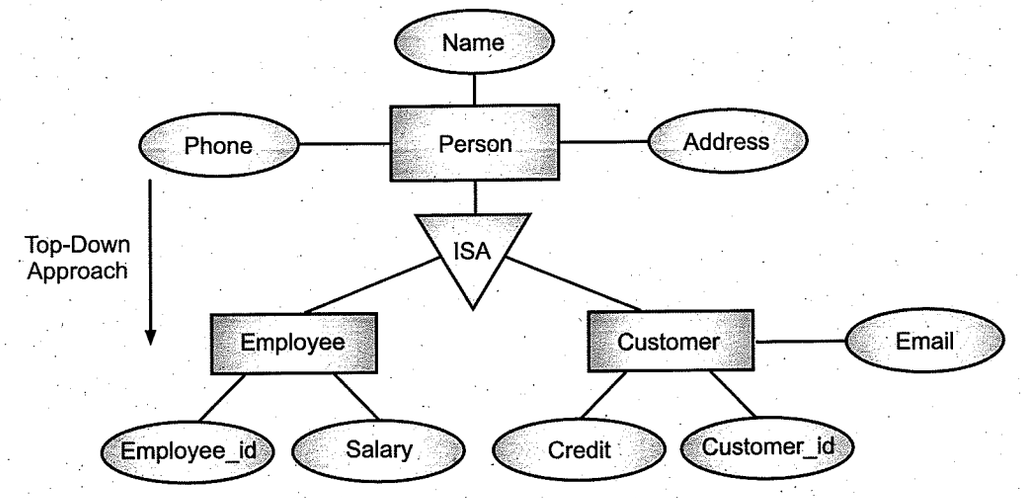


Fig. 2.29: Specialization

- The specialization process allows us to do the following:
  - Define a set of subclasses of an entity type.
  - Establish additional specific attributes with each subclass.
  - Establish additional specific relationship types between each subclass and other entity types or other subclasses.

2.1.7 Aggregation

- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- It helps to express relationship among relationships. To illustrate the need of aggregation consider the database describing information about employees who work on a particular project and use a number of different machines in their work.

- It is the process of combining information on an object so that the higher level objects can be abstracted.
- In any entity relationship diagram, we cannot form a relationship among the entities and their relationship. Aggregation is the process which allows doing so, (See Fig. 2.30).

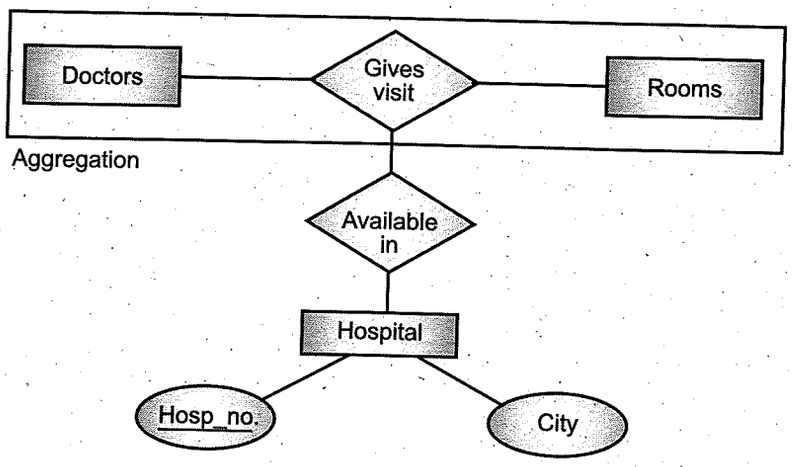


Fig. 2.30: Aggregation

- In above example, many doctors give visits to many rooms and everything is worked into the same hospital.

2.2 RELATIONAL DATA MODEL: STRUCTURE OF RELATIONAL DATABASE MODEL, TYPES OF KEYS

- In this section, we will study Relational model. Database management systems based relational data model are popularly called as RDBMS.
- The relational model was formally introduced by Dr. E. F. Codd in 1970 and has evolved since then, through a series of writings.
- The model provides a simple, yet thoroughly defined, concept of how users identify data.
- The relational model represents data in the form of two-dimension tables. Each table represents some real-world person, place, thing, or event about which information is collected.
- A relational database is a collection of two-dimensional tables.
- The organization of data into relational tables is known as the **logical view** of the database. That is, the form in which a relational database presents data to the user and the programmer.
- The way the database software physically stores the data on a computer disk system is called the **internal view**.

2.2.1 Structure

- In the relational model, a database is a collection of relational tables. A relational table is a flat file composed of a set of named columns and an arbitrary number of unnamed rows.
- The columns of the tables contain information about the table. The rows of the table represent occurrences of the "thing" represented by the table.
- A data value is stored in the intersection of a row and column. Each named column has a domain, which is the set of values that may appear in that column.

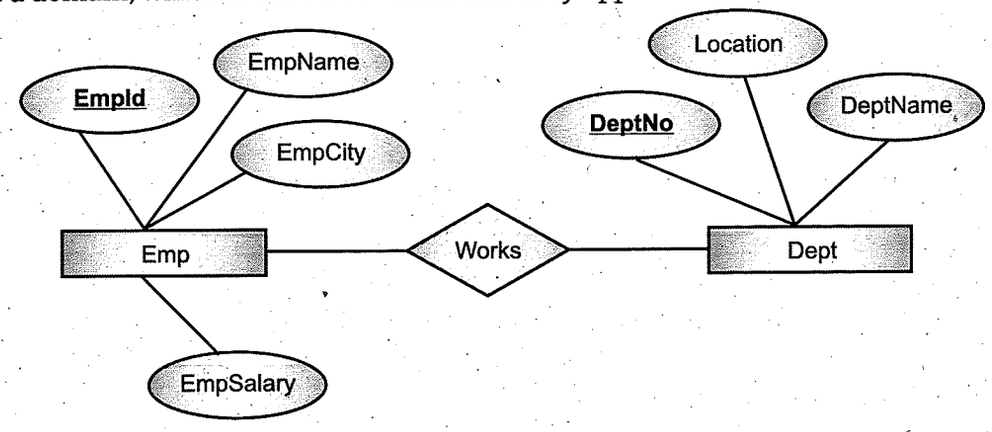


Fig. 2.31: E-R diagram

Table 2.2: Emp Table

EmpID	EmpName	EmpCity	EmpSalary (₹)
1001	Ram	Pune	12,999
1002	Seema	Pune	14,000
1003	Geeta	Delhi	20,000
1004	Johan	Noida	15,000
1005	Ali	Kolkatta	12,999

Table 2.3: Dept Table

Attributes/Columns

EmpID	DeptNo	DeptName	Location
1001	D1	Computer	South block
1002	D2	Electronics	East block
1003	D3	Electrical	Main block
1004	D4	Civil	North block
1005	D1	Computer	West block

Records/ Rows/Tuples

Table 2.4: Works Table

EmpId	DeptNo
1001	D1
1002	D2
1003	D3
1004	D4
1005	D1

- Relational tables can be expressed concisely by eliminating the sample data and showing just the table name and the column names.
- For example: `Emp(Emp_Id, Emp_Name, Emp_City, Emp_Salary)`
  `Dept. (Dept_No, Dept_Name, Location)`
  `Works (Emp_Id, Dept_No)`

Basic concepts of relational model are given below:

1. **Attributes:**
   - Attribute is the name of a column in a table.
   - The attributes of relation Account are acc_no, acc_type, balance. The number of attributes of a relations is called as arity (or degree or order) of the relation. No two attributes of a relation can have same name.
2. **Domain:**
   - Domain (D) is the set of values of the same data type.
   - Domain of an attribute is defined as the set of allowable values for the attribute. Domain is a set having "homogeneous" members and it is conceptually similar to the data type concept in programming languages.
   - According to the set of values, domain is classified into four categories:

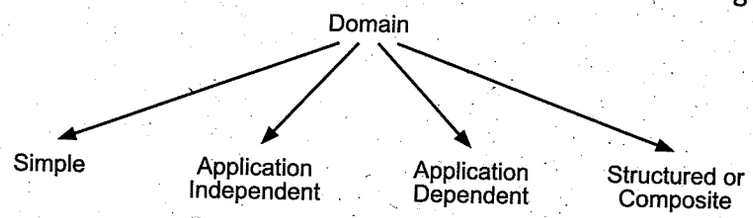


Fig. 2.32: Classification of domain

1. Domain (D) is said to be simple if all its elements are non-decomposable.
2. Application Independent or Atomic domain is the general set of integers, real numbers or character strings.
3. Application Dependent domain is having the values permitted in the database.
4. Structured or Composite domain can be specified as a set consisting of non-atomic values.

- **For example:** Attribute address which specifies street, city. It is possible that several attributes in a relation can have same domain.

3. **Tuple:**
   - Every row of a relation is called as tuple.
   - Tuple is comparable to a record in a conventional file processing systems and an entity in E-R model.
   - Tuple is defined as an ordered set of attributes. Number of tuples in relation is called as **Cardinality** of the relation.
   - Tuples are generally denoted by lower-case letters r, s, t, ... of alphabet. An n-tuple t (having n attributes) can be represented as t (a_1, a_2, ..., a_n) where a_i is a value of i^{th} attribute in tuple t. In tuple representation, the order of attributes is significant and fixed.
4. **Relation:**
   - Relation is a collection of homogeneous tuples.
   - A relation with n attributes is a subset of Cartesian product of domains of those attributes.

     Consider the Account relation in Table 2.5 with attributes:

     `(acc_no, acc_type, balance).`

Table 2.5: Account Table

acc_no	acc_type	balance
31001	saving	10,000
31002	saving	60,000
. . .	. . .	. . .
31009	saving	8,000
31000	saving	4,000

Domain of `acc_no` is `{31001, 31002, 31003, ....., 31049, 31050}`

Domain of `acc_type` is `{saving, current}`

Domain of `Balance` is `{10000, 60000, ....., 4000, 8000}`

- Relation Account is subset of Cartesian Product of domains of all these attributes.
- Relation has two parts:
  (i) Relation schema
  (ii) Tuples

- **A relation has following properties:**
  - (i) The relation has a name that is distinct from all other relation names.
  - (ii) Each cell of the relation contains exactly one atomic single value.
  - (iii) Each attribute has a distinct name.
  - (iv) The values of an attribute are no duplicate tuples.
  - (v) The order of attributes has no significance.
  - (vi) The order of tuples has no significance.

5. Intension and Extension:

- The structure of relation together with a specification of the domains and any other restriction on possible values is called its *intension*, which is usually fixed unless the meaning of the relation is changed to include additional attributes.
- The tuples are called the *extension*(or **state**) of the relation which changes over time.

6. Degree:

- The degree of a relation is the number of attributes it contains.
- A relation with only one attribute would have a degree of one is called Unary Relation.
- A relation with two attributes is called a Binary Relation and with n attributes is called an n-ary relation.

7. Cardinality:

- The cardinality of relation is the number of tuples it contains.
- The cardinality is the property of the extension of the relation and is determined from the particular instance of the relation at any given moment.

8. Relation Schema:

- It defines the set of attributes of that relation. It can be represented as R-schema $(A_1, A_2, A_3, ..., A_n)$ where the domain of each attribute A_i is D_i and the relation R over the set of attributes.

  R-schema is denoted as R (R-schema) and R (R-schema) $\subseteq D_1 \times D_2 \times D_3, ..., \times D_n$.
- The Account relation in Table 2.6 can be represented as Account (Account_schema), where Account_schema (acc_no, acc_type, balance).

9. Relational Database:

- A collection of normalized relations with distinct relation names is called as Relational Database.

10. Relational Database Schema:

- A set of relation schemas, each with distinct name is called as Relational database Schema.

11. NULL Value:

- Null Value indicates that the value for corresponding attribute is either not available or not applicable.
- It is distinct from empty character string and it is distinct from zero or any other number.

2.2.2 Types of Keys

Keys of RDBMS:

- Keys are fundamental to relational databases. Relational database will not be usable without keys.
- Every table must have some columns or combination of columns which uniquely identify each row in the table. For that we require a key. A key is simply a field used to identify a record. In other words, a key is relation of subset of attributes with following properties:
  1. The value of key is unique for each tuple.
  2. No data redundancy.
- Key can be classified as:
  1. Primary key
  2. Candidate key
  3. Foreign key
  4. Super key
  5. Composite key
  6. Secondary Key

1. Primary Key:

- In a relation, there is one attribute or a group of attributes with values that are unique within that relation and thus can be used to uniquely identify the tuples of that relation.
- This key which uniquely identifies a row is known as **Primary key** of that relation.
- For example, consider a Customer table, where social security number of the entity set customer is sufficient to distinguish one customer entity from another. So *social security number* is a primary key.

Table 2.6: Primary key of table 'Customer'

Social_Security No.	Cust_name	Address
100 – 205	John	Paris
103 – 102	Jimmy	New York
208 – 56	Juliee	Rye

2. Candidate Key:

- The attribute which possess the unique identification property in a relation is called as **candidate key**. There can be more than one candidate keys in a relation.

OR

- A table can have more than one set of columns that could be chosen as the key called **candidate key**.
- In other words, a set of attributes that uniquely identifies a tuple according to a key which is **minimal** set of the columns in the table.
- Candidate key is defined by two parts:
  (i) Two distinct tuple (record) cannot have same value in all the fields of a key.
  (ii) No subset of the set of fields (attributes) in a key is a unique identifier or a tuple.
- Candidate key should have following things:
  (i) It must be unique.
  (ii) A candidate key's value must exist. It cannot be null.
  (iii) The value of the candidate key must be stable. Its value cannot change outside the control of the system.
- For example, in customer table, various employees were working and having unique identification number called as Social Security Number (SSN).

Table 2.7: Candidate key of Customer table

Customer name	SSN	Basic Sal
John	001-256	₹ 14,000
Martin	005-123	₹ 2,000
Paster	008-200	₹ 1,000
Mary	101-401	₹ 18,000
Johns	102-030	₹ 48,000

Candidate key

3. Foreign Key:

- If any key in a given relation has reference to the value of a primary key of some other relation then it is called as **Foreign** key.
- Foreign key can have duplicate values it is used to search a record from two relations.

Relation = Department

Primary Key

Table 2.8: Department Table

dept_no	Name
1	Production
2	Purchasing
3	Marketing

Relation = Employee

Primary Key

Foreign Key

Table 2.9: Employee Table

emp_id	Emp_name	Dept._rw
10	Ramesh	2
25	Indranil	1

- A foreign key is a set of columns in one table, which is a key in another table.
- Foreign keys are very important in DBMS to show the relationship between tables.
- **Example:** Consider the following two tables: Owner and Car.

```
Owner (Licence_no, name, addr, phone)
Car (Car_no, model, colour)
```

- Owner and Car are related with One-to-Many relationship.
- The primary key of relation one (called parent) is taken and added as a foreign key into the many (called child) relation. Therefore, to show the One-to-Many relationship, we add **licence_no** in the table Car.

```
Owner (Licence_no, name, addr, phone)
Car (Car_no, model, colour, licence_no)
```

Foreign key

- **Example:** Consider two tables: student and class. Each student enrolled in a class. One class can have many students. The relationship is One-to-Many.

```
Class (class_id, division)
Student (stud_id, name, age, addr)
```

- We can add **class_id** (primary key of class table) in the student table as foreign key.

4. Super Key:

- Super key is a set of one or more attributes which taken collectively allows us to identify uniquely an entity in the entity set.

- Consider the entity set account with attributes {acc_no., cust_name, bank_name, balance, address}.
- Here, acc_no is super key of account entity set. Similarly, (acc_no, cust_name) is a super key and (acc_no., bank_name) is a super key. But cust_name and bank_name independently are not the super keys.
- Super key may contain extra attributes i.e. if k is a super key, then any super set of k is also a Super key.

5. Composite Key:

- There are situations when one attribute cannot form a key, because single attribute cannot uniquely identify every tuple or row in the table. So we need two or more attributes to identify tuple in the table.
- The key which consists of two or more columns is called as **composite key**.
- **For example:** Consider a table of supplier which tells us the parts which supplier sells. Table is shown below:

Table 2.10: Supplier table

Sup_id	Part_id	Quantity	Price
S1	P1	12	300
S1	P2	10	100
S2	P2	5	50
S3	P3	7	200

- Here, neither the Sup_id nor the Part_id can identify a row in the table uniquely. However, two of them together can easily identify a tuple or row in the table uniquely. Hence, composite key is {Sup_id, Part_id}.

6. Alternate Key or Secondary Key:

- Alternate keys or Secondary keys are keys that may or may not identify a record uniquely, but helps in faster scanning.
- In other words, Alternate key is any candidate key which is not the primary key.

Significance of Alternate Key:

(i) The DBMS creates and uses an index based on the alternate key, like primary key.

(ii) Searching is faster because it is index.

(iii) Alternate key can have duplicates.

2.2.3 Referential Integrity Constraints

- Relational Model Constraints is a mechanism used to prevent invalid data entry into the table.
- An integrity constraint means that you are enforcing some conditions on the attribute.

- The types of constraints are:
  1. Domain Integrity Constraints
  2. Entity Integrity Constraints
  3. Referential Integrity Constraints
  4. On Delete Cascade.

1. Domain Integrity Constraints:

- It is used to maintain the value according to the user specifications. There are basically two types of domain constraints:

(i) **Not null constraints:** By default all columns in a table allows NULL values. When a 'not null' constraint is enforced on a column or a set of columns, it will not allow NULL values.

Syntax and Example:

```
Create table Emp(EmpId number(4), EmpName varchar2(20) not null, phoneno number(10));
```

- After table creation *not null* can be added as:

```
Alter table Emp modify EmpName varchar2(20) not null;
```

(ii) **Check constraints:** This constraint defines a condition that each row must satisfy. A single column can have multiple check conditions.

Syntax:

```
Create table tablename(column1 datatype size constraint
   Constraint_name primary key……);
```

Where, constraint and primary key are the keywords and constraint_name is any user enter name.

Example:

```
Create table Emp(EmpId number(4), EmpName varchar2(20) not null, salary number(10, 2) constraint chk_emp check(salary is not null and salary>20000), phoneno number(10));
```

If user tries to enter the salary less than 20000 it will give an error.

- After table creation *check constraint* can be added as:

```
Alter table Emp add constraint chk_emp check(salary > 20000);
```

2. Entity Integrity Constraints:

- An entity is any data recorded in database. Each database represents a table and each row of a table represents an instance of that entity.
- There are two types of entity integrity constraints:

(i) Primary Key Constraint (Integrity Rule 1):

- It is used to prevent the duplication of values within the rows of a specified column or set of columns in a table. It does not allow NULL values. If primary key constraint is defined to more than one column it is said to be composite primary key.

Syntax:

```
Create table tablename(column1 datatype size constraint
                                    constraint_name primary key……);
```

Example:

```
Create table Emp(EmpId number(4) constraint emp_pk primary key, EmpName
varchar2(20) not null, salary number(10, 2) constraint chk_emp check(salary
is not null and salary > 20000), phoneno number(10));
```

- After table creation the primary key is added as:

```
Alter table Emp add constraint chk_emp primary key(EmpId);
```

(ii) Unique Constraint:

- It is used to prevent the duplication of values within the rows of a specified column or set of columns in a table. It allows NULL values. If unique constraint is defined to more than one column it is said to be composite unique key.

Syntax:

```
create table tablename(column datatype size constraint
                                    constraint_name unique……);
```

Example:

```
Create table Emp(EmpId number(4), EmpName varchar2(20) not null, salary
number(10,2) constraint chk_emp check(salary is not null and salary>20000),
phoneno number(10) constraint ph_uk unique);
```

- Suppose you want to apply unique constraints to more than one column:

```
Create table Emp(EmpId number(4), EmpName varchar2(20) not null, salary
number(10,2) constraint chk_emp check (salary is not null and
salary>20000), deptno number (5), phoneno number(10), constraint ph_uk
unique(deptno,phoneno);
```

- After table creation the primary key is added as:

```
Alter table Emp add constraint ph_uk unique(phoneno);
```

3. Referential Integrity Constraint:

- It is used to establish a parent child relationship between two tables. A value of foreign key is derived from the primary key. Primary key is defined in a parent table and foreign key is defined in child table. The child table contains the values for foreign key column which are present in parent tables primary key column but not other than that.

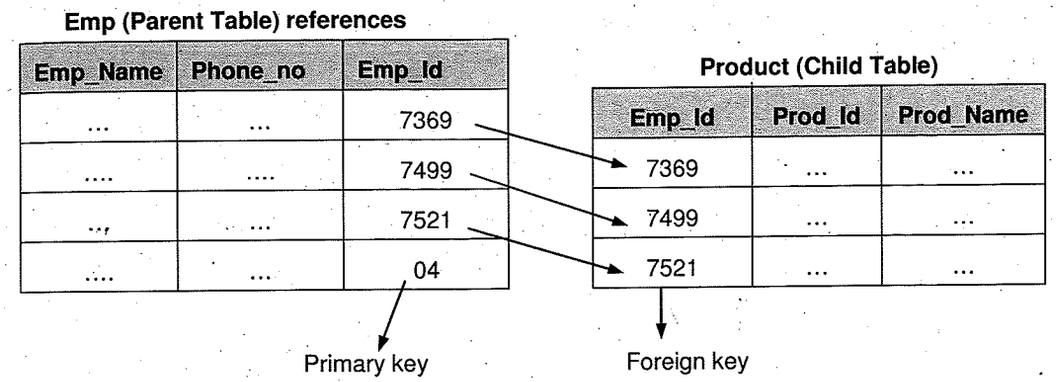


Fig. 2.33: Parent-child relationship between two tables

Syntax:

```
Create table tablename(column datatype size references
                                    parenttablename(primary key attribute)……);
```

Example:

```
Create table Product(Emp_Id number(4) references Emp(Emp_Id),
                                    P_Name varchar2(10));
```

- After table creation the foreign key is added as:

```
Alter table Product add constraint fk_prod foreign key(Emp_Id)
                                    references Emp(Emp_Id);
```

2.3 CODD'S RULES

- In 1985 E. F. Codd proposed an informal set of twelve rules by which a database could be evaluated to see how relational it is.
- Very few commercial databases exist which meet or satisfy all twelve rules. All twelve rules are built on the following foundation rule, Rule 0.

Rule 0: The Foundation rule

- Any truly relational database must be manageable entirely through its own relational capabilities.

Rule 1: The Information Rule

- All information in a relational database is represented explicitly at the logical level and in exactly one way – by values in tables.
- In simple terms, if an item of data doesn't reside in a table in the database then it doesn't exist. This can should be extended to the point where even such information as table, view and column etc. should be contained somewhere in table form.

Rule 2: The Rule of Guaranteed Access

- Each and every datum (atomic value) in relational database is guaranteed to be logically accessible by restoring to a combination of table name, primary key value and column value.

- If a database conforms to Rule 2, every atomic value should be easily retrievable.

Rule 3: The Systematic Treatment of NULL Values

- Null Values are supported in a fully relational DBMS for representing missing information in a systematic way, independent of data type.

Rule 4: The Database Description Rule

- The description of the database is held and maintained using the same logical structures used to define the data, thus allowing users with appropriate authority to query such information in the same ways and using the same language as they would query any other data in database.
- Rule 4 states that there must be a data dictionary within the RDBMS that is constructed of tables and/or views that can be examined using SQL.

Rule 5: The Comprehensive Sub Language Rule

- There must be at least one language whose statements can be expressed as character strings conforming to some well defined syntax, that is comprehensive in supporting the following:
  - Data definition.
  - View definition.
  - Data manipulation.
  - Integrity constraint.
  - Authorization.
  - Transaction boundaries.
- This means that the RDBMS must be completely manageable through its own dialect of SQL.

Rule 6: The View Updating Rule

- All views that can be updated in theory can be updated by the system.

Rule 7: The Insert and Update Rule

- The capability of handling a base relation, or in fact a derived relation, as a single operand must hold good for all retrieve, update, delete, and insert activity.

Rule 8: The Physical Independence Rule

- User access to the database, via terminal monitors or application programs, must remain logically consistent whenever changes to the storage representation, or access methods to the data, are changed.

Rule 9: The Logical Data Independence Rule

- Application programs and terminal activities must remain logically unimpaired whenever information preserving changes of any kind that are theoretically permitted, are made to the base table.

Rule 10: Integrity Independence Rule

- All integrity constraints defined for a database must be definable in the language referred to in Rule 5 and stored in the database as data in tables.
- The following integrity rules should apply to every relational database:
  - **Entity Integrity:** No component of a primary key can have missing values.
  - **Referential Integrity:** For each distinct foreign key value there must exist a matching primary key value in the same domain.

Rule 11: Distribution Independence Rule

- An RDBMS must have distribution independence. It states that applications running on a non-distributed database must remain logically unimpaired that data if that data should then become distributed in the context of a distributed relational database.

Rule 12: No Subversion Rule

- If an RDBMS supports a lower level language that permits (for example row-at-a-time) processing, then this language must not be able to bypass any integrity rules or constraints defined in the higher level, set at a time, relational language.

2.4 DATABASE DESIGN USING E-R, E-R TO RELATIONAL

- Database design is the organization of data according to a database model. The designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to the database model. Database management system manages the data accordingly.

2.4.1 Database Design using E-R

Entity Relationship Diagram:

- Entity Relationship Diagram, also known as ERD, E-R Diagram or E-R model, is a type of structural diagram for use in database design.
- An ERD contains different symbols and connectors that visualize two important things: **The major entities within the system scope**, and the **inter-relationships among these entities**.
- Entity Relationship (E–R) modelling is a design tool. It is a graphical representation of the database system which provides a high-level conceptual data model and supports the user's perception of the data.
- First step of any relational database design is to make E-R Diagram for it and then convert it into relational model. Relational Model represents how data is stored in database in the form of table.
- Following are the symbols used in E-R diagram:

Table 2.11: Symbols of E-R diagram

Sr. No.	Symbol Name	Description	Symbol
1.	Rectangles	Represent entity	
2.	Ellipses	Represent attributes.	
3.	Diamonds	Represent relationship set.	
4.	Lines	Link attributes to entity sets and relationship sets.	
5.	Double ellipse	Represent multi-valued attribute.	
6.	Dashed ellipse	Represent derived attribute.	
7.	Double lines	Indicate total participation of an entity in a relationship set.	
8.	Double rectangle	Represent weak entity set.	

1. Entities:

- An entity is any object in the system that we want to model and store information about database.
- Individual objects are called as entities.
- Groups of the same type of objects are called entity types or entity sets.
- Entities are represented by rectangles as shown in following Fig.

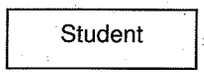


Fig. 2.34: Entity 'Student'

- There are two types of entities: Weak entity and Strong entity.

2. Attribute:

- All the data relating to an entity is held in its attributes.
- An attribute is a property of an entity.
- Each attribute can have any value from its domain.
- Each entity within an entity type:
  - May have any number of attributes.
  - Can have different attribute values than that in any other entity.
  - Have the same number of attributes.

3. Relationships:

- A relationship type is a meaningful association between entity types.
- A relationship is an association of entities where the association includes one entity from each participating entity type.
- Relationship types are represented on the E–R diagram by a series of lines.
- The relationship is placed inside a diamond symbol.

How to draw E-R diagram (ERD)?

- Make sure you are clear about the **purpose** of drawing the ERD. Are you trying to present an overall system architecture which involves the definition of business objects? Or are you developing an E-R model ready for database creation? You must be clear about the purpose to develop an E-R diagram at right level of detail.
- Make sure you are clear about **the scope** to model. Knowing the modelling scope prevents you from including redundant entities and relationships in your design.
- Draw the major **entities** involved in the scope.
- Define the **properties** of entities by adding columns.
- Review the ERD carefully and check if the entities and columns are enough to store the data of the system. If not, consider adding additional entities and columns. Usually, you can identify some transactional, operational and event entities in this step.
- Consider the **relationships between all entities and relate** them with proper **cardinality** (e.g. One-to-Many relation between entity Customer and Order).
- Apply the technique of **database normalization** to restructure the entities in a way that can reduce data redundancy and improve data integrity. For example, the details of manufacturer might be stored under the Product entity initially. During the process of normalization, you may find that the detail keeps repeating record over record, then you can split it as a separate entity Manufacturer, and with a foreign key that links between Product and Manufacturer.

SOLVED EXAMPLES

Example 1: Draw an E-R diagram for order processing system where a person can give order for many items by specifying its quantity.

Solution:

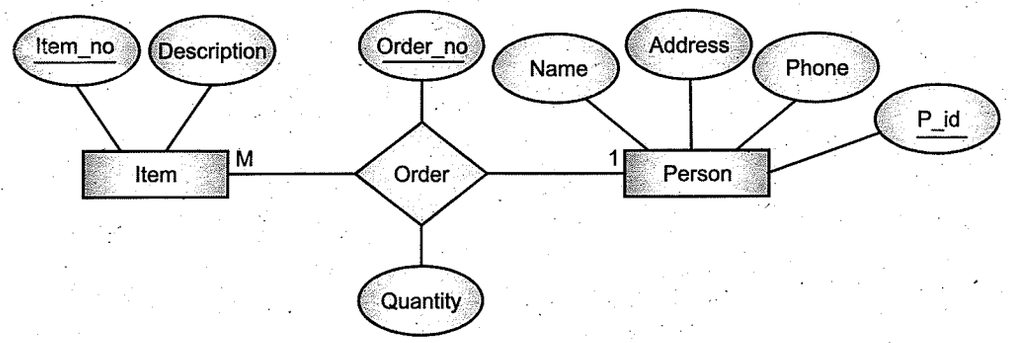


Fig. 2.35: E-R Diagram for Order Processing System

Example 2: Draw an E-R diagram for Airlines Reservation System. Here, a passenger can book ticket from personal for a flight on some date.

Solution:

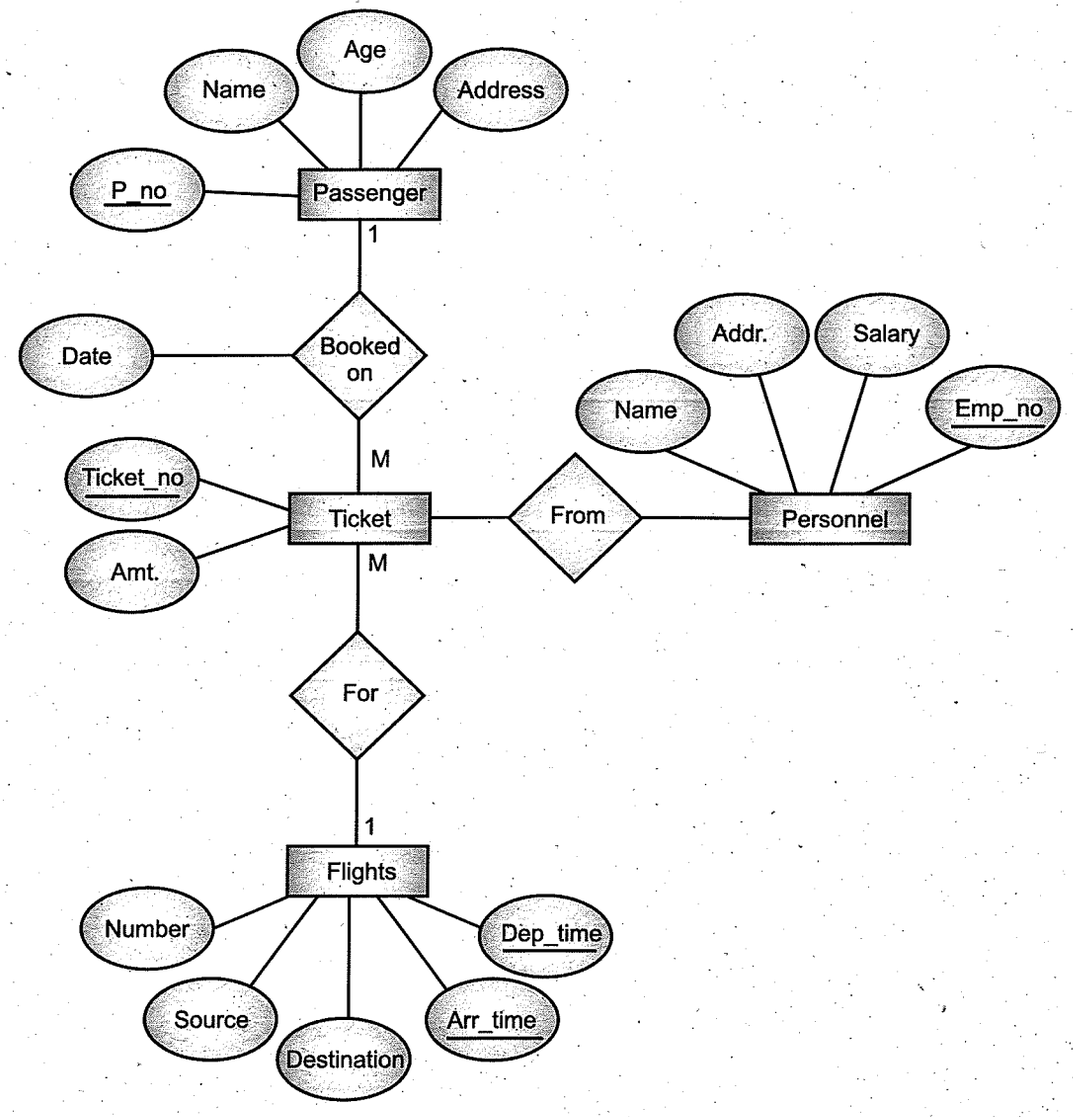


Fig. 2.36: E-R Diagram for Airline Reservation System

Example 3: Construct an E-R diagram for a car insurance company that has a set of customers. Each customer owns one or more cars. Each are has associated with more cars. Each car has associated with zero to any number of recorded accidents.

Solution:

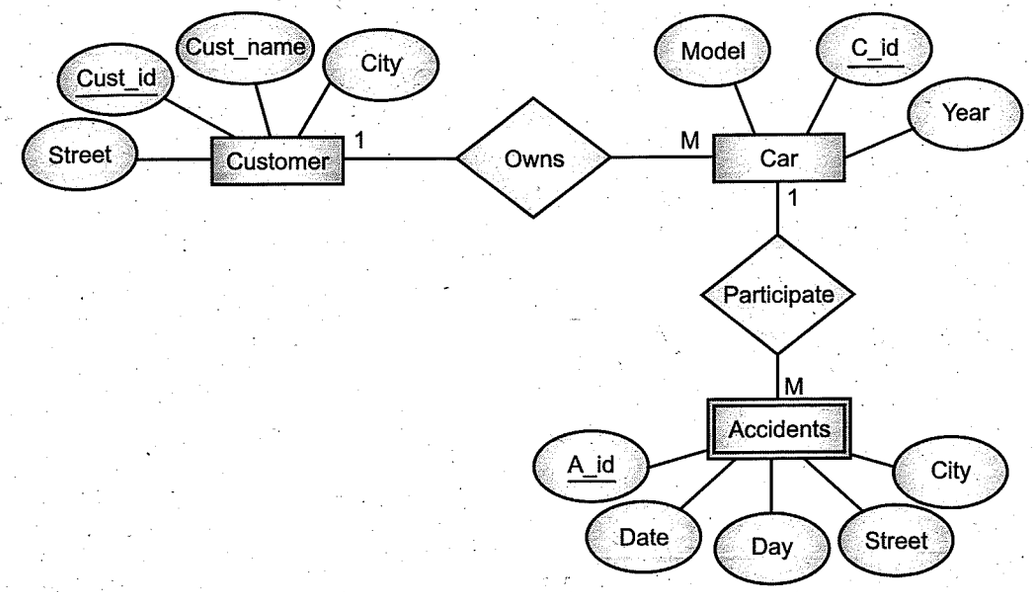


Fig. 2.37: E-R Diagram for Car Insurance System

Example 4: A movie studio wishes to institute a database to manage their files of movies, actors and directors. The following facts are relevant.

(i) Each actor has appeared in many movies.

(ii) Each director has directed many movies.

(iii) Each movie has one director and one or more actors.

(iv) Each actor and director may have several addresses.

Draw E-R diagram.

Solution:

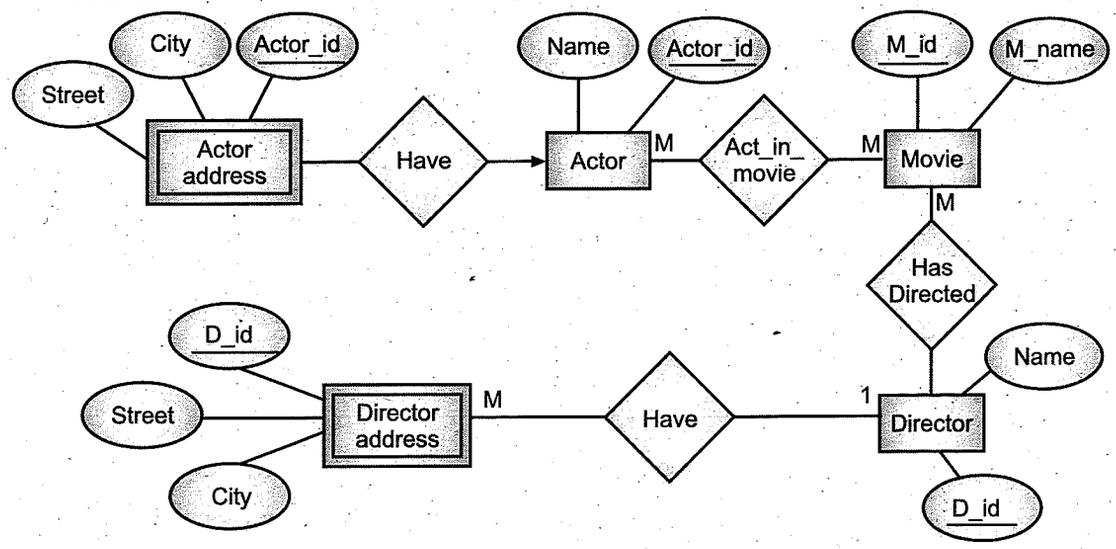


Fig. 2.38: E-R Diagram for Film Industry System

Example 5: Following information is maintained manually in a library.

Books (Access_number, name, authors, price, book_type, publisher)

Borrowers (membership_no., name, address, category, max_no of books that can be issued, Access_number of books borrowed)

The following constraints are observed:

1. Each book has unique access-number.
2. A book may have more than one author.
3. There may be more than one copy of a book.
4. The category of borrower determines the maximum number of books that may be issued to borrower.

- Identify the entities, relationship and draw E-R diagram.
- Provide for issue and return of book, fine calculation and claiming of issued book.

Solution:

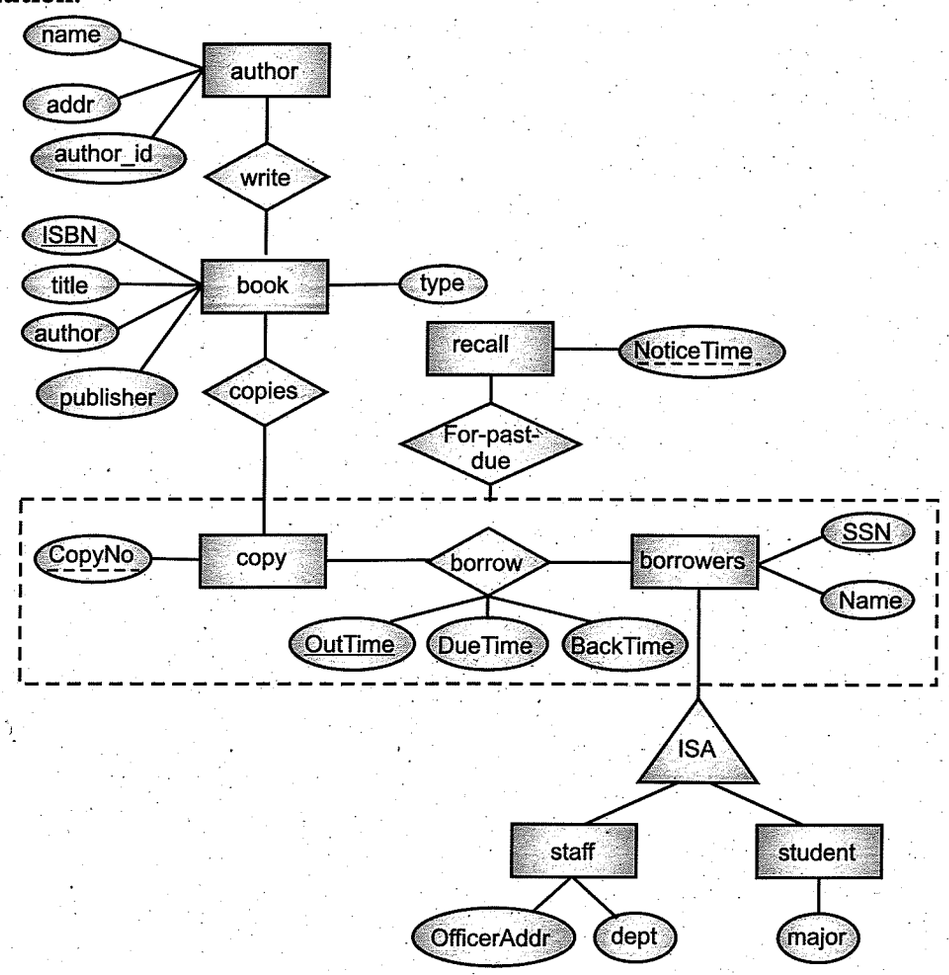


Fig. 2.39: E-R diagram for Library System

Example 6: Consider the following information about a university database.

- Professors have an id, a name, an age, a rank and a research area.
- Projects have a project number, a funding agency, a starting date, and finish date and a budget.
- Under-graduate students have an id, a name, an age, a course.
- Each project is managed by one professor, called as Principal Investigator.
- One or more professors, called as Co-investigators, work each project on.
- When graduate students work on a project, a professor must supervise their work on the project.
- Graduate students can work on multiple projects in which case they will have different supervisor for each one.
- Departments have a department number, department name and an office.
- Departments have a professor known as HOD.

Design and draw the E-R diagram.

Solution:

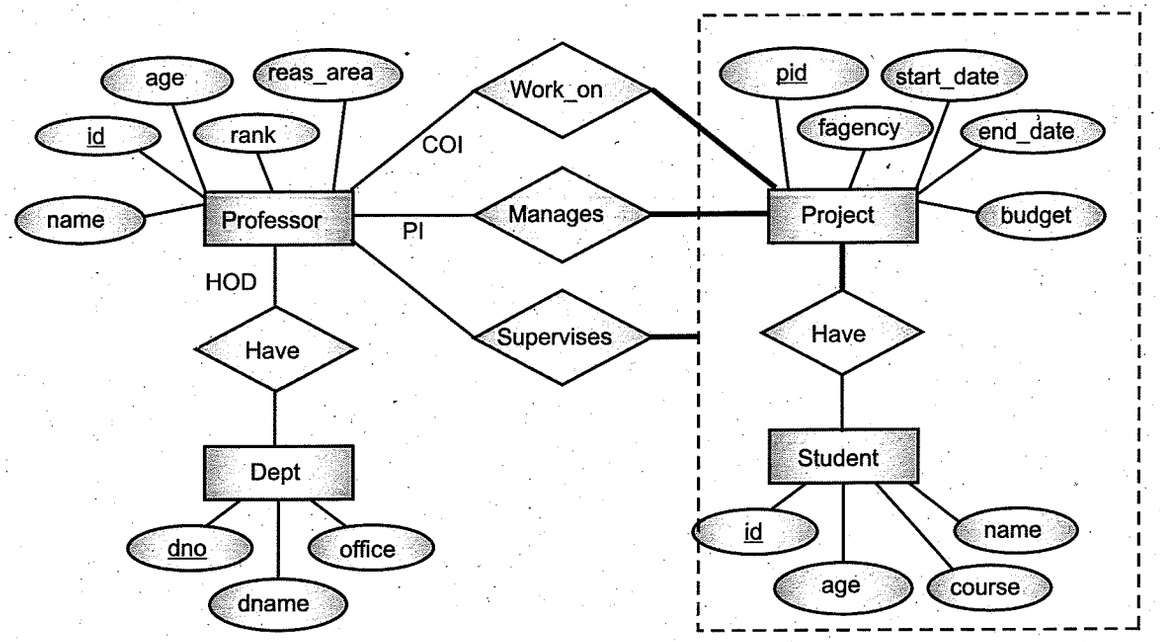


Fig. 2.40: E-R diagram for University database

2.4.2 E-R Diagram to Relational

- An E-R model can be converted to relations, in which each entity set and each relationship set is converted to a relation. Fig. 6.12 illustrates a conversion of E-R diagram into a set of relations.

- Let's learn step by step how to convert E-R diagram into relational model:

1. Entity Set:

- Consider we have entity STUDENT in E-R diagram with attributes RollNumber, Student Name and Class.
- To convert this entity set into relational schema:
  1. Entity is mapped as relation in Relational schema.
  2. Attributes of Entity set are mapped as attributes for that Relation.
  3. Key attribute of Entity becomes Primary key for that Relation.

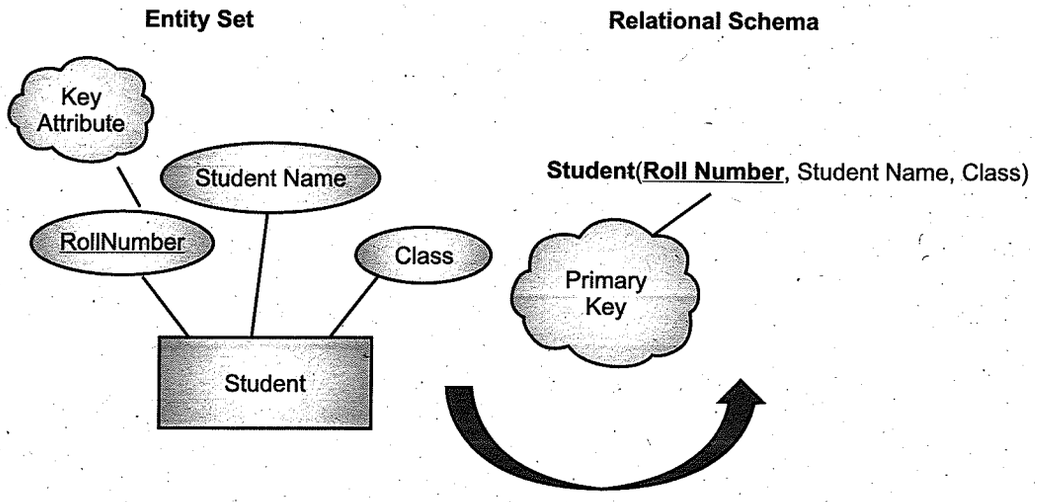


Fig. 2.41: Key attribute shown as Primary Key

2. Entity set with multivalued attribute:

- Consider we have entity set Employee with attributes *Employee ID, Name* and *Contact number.*
- Here *contact number* is multivalued attribute as it has multiple values. As an employee can have more than one contact number for that we have to repeat all attributes for every new contact number. This will lead to data redundancy in table. Hence to convert entity with multivalued attribute into relational schema, separate relation is created for multivalued attribute in which:
  1. Key attribute and multivalued attribute of entity set becomes primary key of relation.
  2. Separate relation employee is created with remaining attributes. Due to this instead of repeating all attributes of entity now only one attribute is need to repeat.

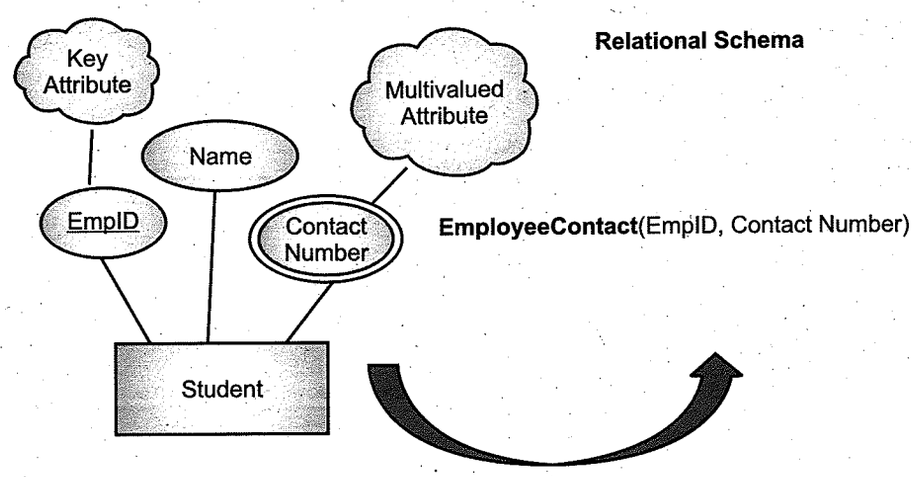


Fig. 2.42: Entity Set with Multivalued Attribute

3. Entity set with Composite attribute:

- Consider entity set Student with attributes Roll Number, Student Name and Class. Here Student Name is composite attribute as it has further divided into First Name, Last Name.
- In this case, to convert entity into relational schema, composite attribute Student Name should not be include in relation but all parts of composite attribute are mapped as simple attributes for relation.

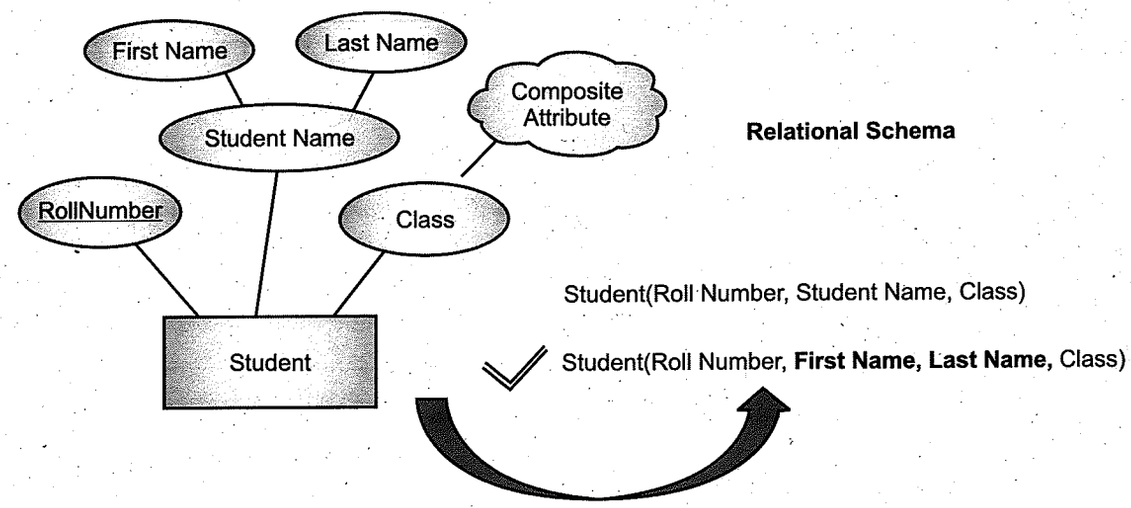


Fig. 2.43: Entity set with Composite attribute

4. 1:M (One to Many) Relationship:

- Consider 1:M relationship set enrolled exist between entity sets Student and Course as following:

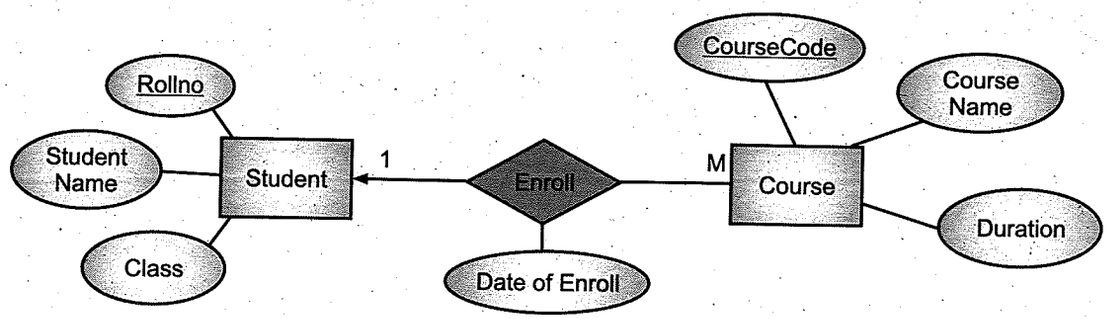


Fig. 2.44(a): 1: M Relationship

- Attributes of entity set Student are Rollno (which is primary key), Student Name and Class.
- Attributes of entity set Course are CourseCode (which is primary key), Course name and Duration and Date of Enroll is attribute of relationship set Enroll.
- Here Enroll is 1:M relationship exist between entity set Student and Course which means that one student can enroll in multiple courses.
- In this case to convert this relationship into relational schema,
  1. Separate relation is created for all participating entity sets (Student and Course).
  2. Key attribute of *Many* side entity set (Course) is mapped as foreign key in *One*'s side relation (Student).
  3. All attributes of relationship set are mapped as attributes for relation of one's side entity set (Student).

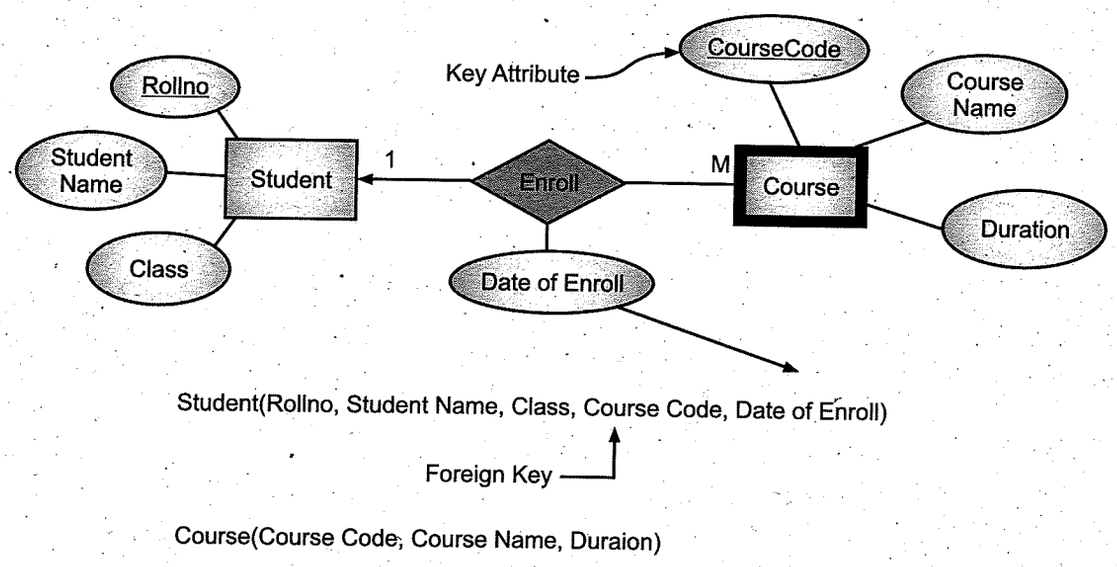


Fig. 2.44(b): 1:M Relationship Schema

5. M:1 (Many to One) Relationship:

- Consider same relationship set Enroll exist between entity sets Student and Course. But here Student is *Many* side entity set while Course is *One*'s side entity set. That means many students can enroll in one course.

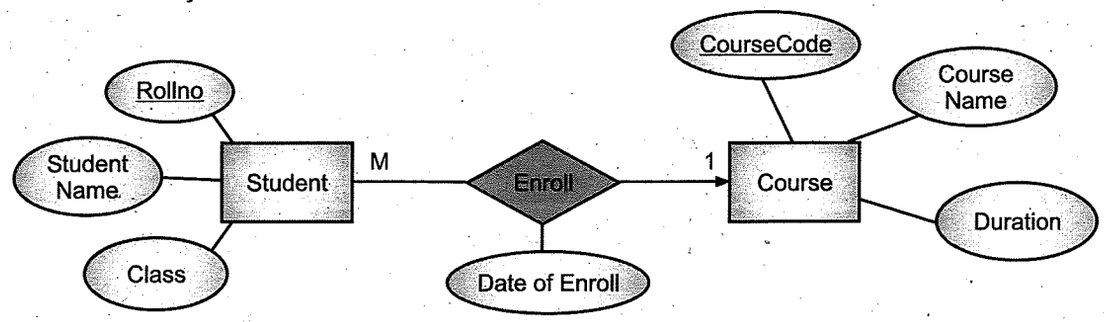


Fig. 2.45(a): M:1 Relationship

- To convert this relationship set into relational schema,
  1. Separate relation is created for all participating entity sets.
  2. Key attribute of *Many* side entity set Student is mapped as foreign key in *One's* side relation.
  3. All attributes of relationship set are mapped as attributes for *One*'s side relation course.

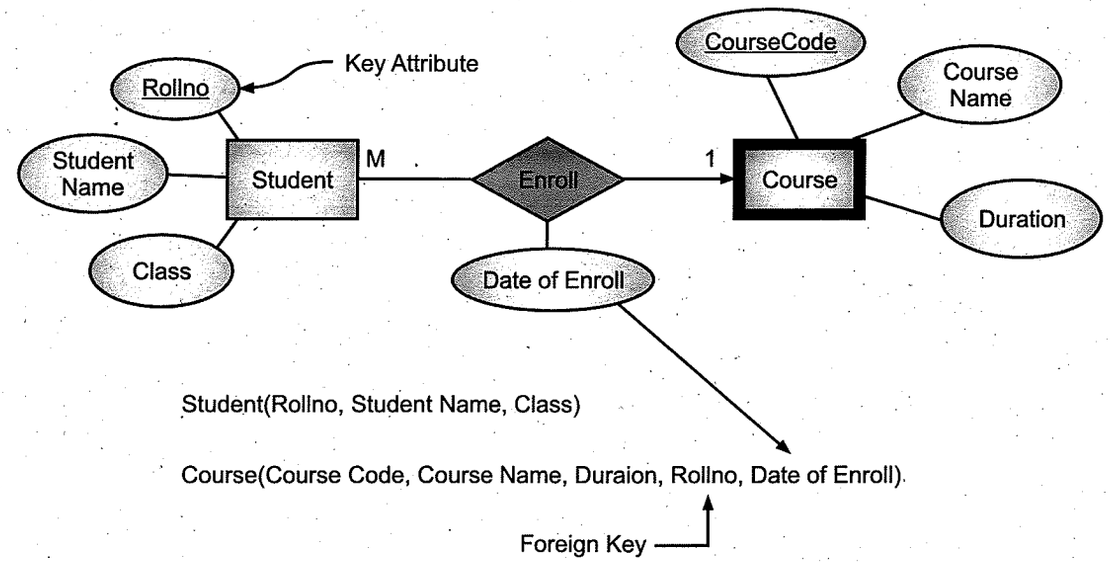


Fig. 2.45(b): M:1 Relationship Schema

6. M:N (many to many) Relationship:

- Consider same relationship set enrolled exist between entity sets Student and Course , which means multiple students can Enroll in multiple courses.

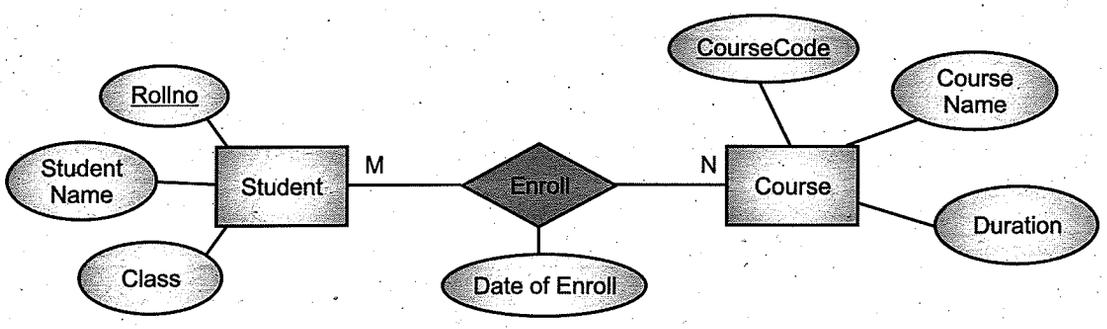


Fig. 2.46(a): M:N Relationship

- To convert this Relationship set into relational schema,
  1. Relationship set is mapped as separate relation.
  2. Key attributes of participating entity sets are mapped as primary key for that relation.
  3. Attribute of relationship set becomes simple attributes for that relation.
  4. And separate relation is created for other participating entities.

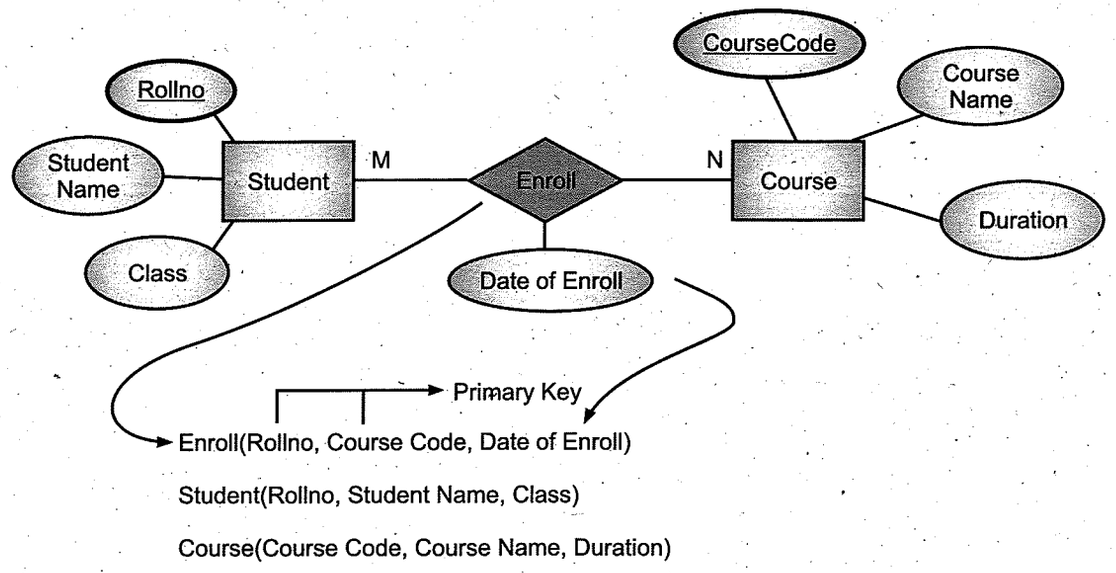


Fig. 2.46(b): M:N Relationship Schema

7. 1:1 (One to One) Relationship:

- Consider same relationship set Enroll exist between entity sets Student and Course, which means one student can Enroll in only one course.

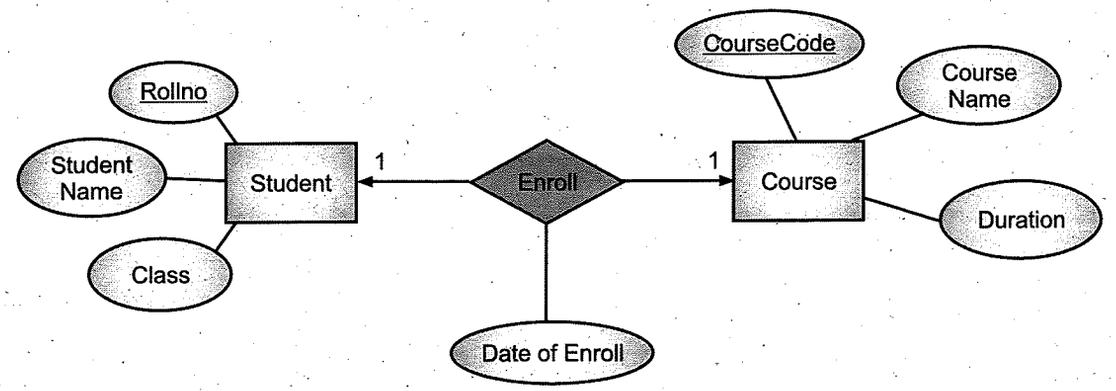


Fig. 2.47 (a): 1:1 Relationship

- To convert this Relationship set into relational schema,
  1. Separate relation is created for all participating entity sets.
  2. Primary Key of Relation Student can be act as foreign key for relation Course OR Primary Key of Relation Course acts as foreign key for relation Student.

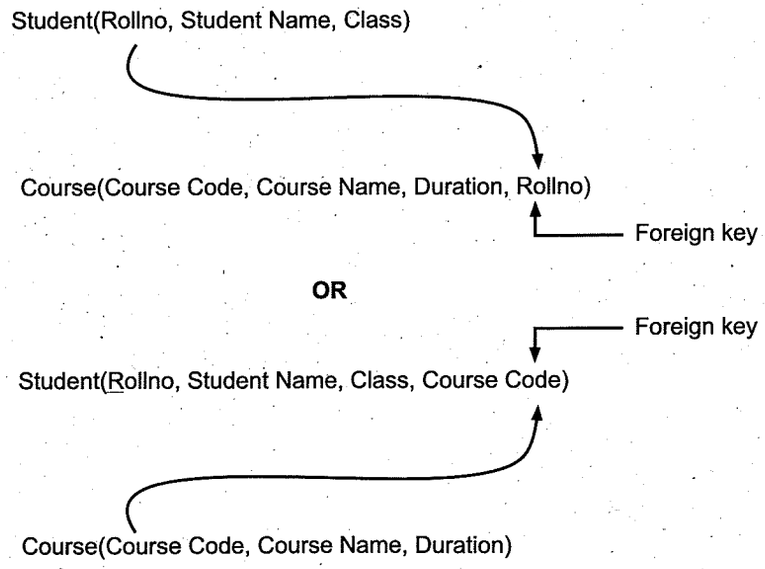
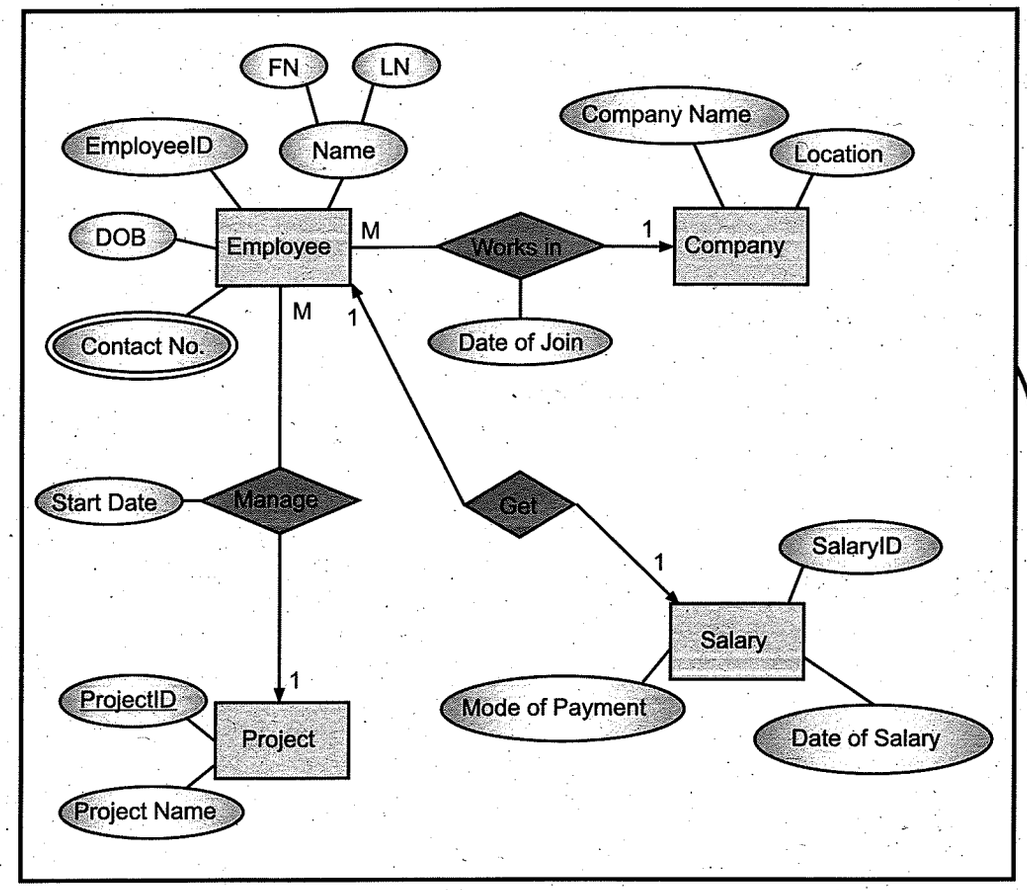


Fig. 2.47 (b): 1:1 Relationship Schema

Solved Examples

Example 7: Convert given ER diagram into Relational Model



ER Diagram

Solution:

Employee(EmployeeID, DOB, FN, LN, Salary ID)

EmployeeContact(EmployeeID, Contact No)

Company(Company Name, Location, Employee ID, Date of Join)

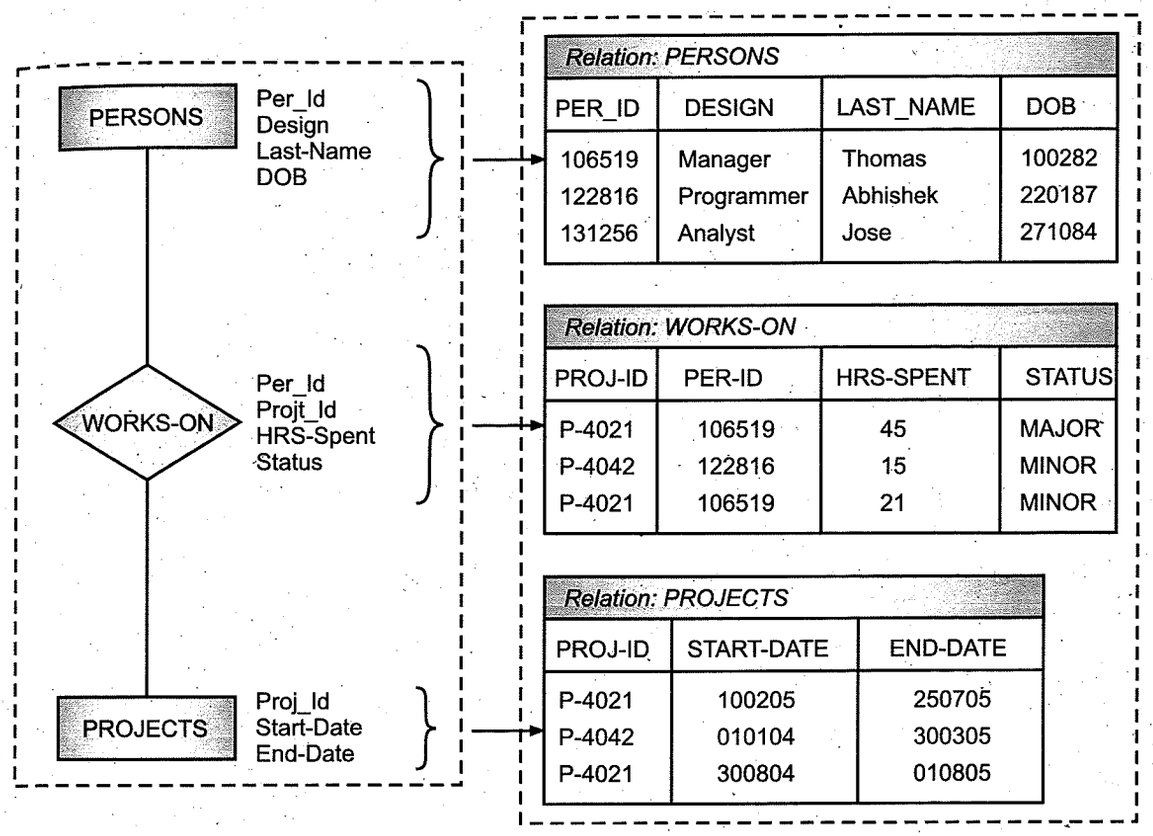
Project(ProjectID, Project Name, Employee ID, Start Date)

Salary (Salary ID, Mode of Payment, Date of Salary), Employee ID)

Relational Model

Fig. 2.48

Example 8: Convert given E-R model into Relational databases



Relation: PERSONS

PER_ID	DESIGN	LAST_NAME	DOB
106519	Manager	Thomas	100282
122816	Programmer	Abhishek	220187
131256	Analyst	Jose	271084

Relation: WORKS-ON

PROJ-ID	PER-ID	HRS-SPENT	STATUS
P-4021	106519	45	MAJOR
P-4042	122816	15	MINOR
P-4021	106519	21	MINOR

Relation: PROJECTS

PROJ-ID	START-DATE	END-DATE
P-4021	100205	250705
P-4042	010104	300305
P-4021	300804	010805

(a) E-R Model **(b) Converted relations**

Fig. 2.49

Solution: Fig. 2.49 (a) is converted to the following three relations as shown in Fig. 2.49 (b) and below in table format:

Table 2.12: Converted relation sets

PERSONS	(PER_ID, DESIGN, LAST-NAME, DOB) from entity set PERSONS
PROJECTS	(PROJ_ID, START-DATE, END-DATE) from entity set PROJECTS
WORKS-ON	(PROJ_ID, PER_ID, HRS-SPENT, STATUS) from relationship set WORKS-ON

2.5 NORMALIZATION – NORMAL FORMS BASED ON PRIMARY (1 NF, 2 NF, 3NF, BCNF)

- Normalization is a process of refining database structures to improve the speed at which data can be accessed and to increase database integrity.
- The tables and fields you define make up the structure of the database. The structure you decide upon affects the performance of your database programs. Some database layouts can improve access speed.
- For example, placing all related information in a single table allows your programs to locate all needed data by looking in one place. On the other hand, you can layout your database in a way that improves data integrity. For example, placing all the invoice line item data in one table and the invoice address information in another table prevents users from deleting complete addresses when they remove invoice line items from the database.
- The repetitive fields make it easier to load large amounts of data in a single SQL statement instead of finding additional, related data in subsidiary tables linked through those slower index constraints.
- Databases built for maximum integrity have many small data tables. Each of these tables can have several indexes mostly foreign keys referencing other tables in the database.
- Well-normalized tables are easy to understand when you look at them. It is easy to see what kind of data they store and what types of updates need to be performed. It is rather easy to create data entry routines and simple reports directly from well-normalized tables.
- In fact, the rule of thumb is this: If it's hard to work with a data table, it probably needs more normalization work.

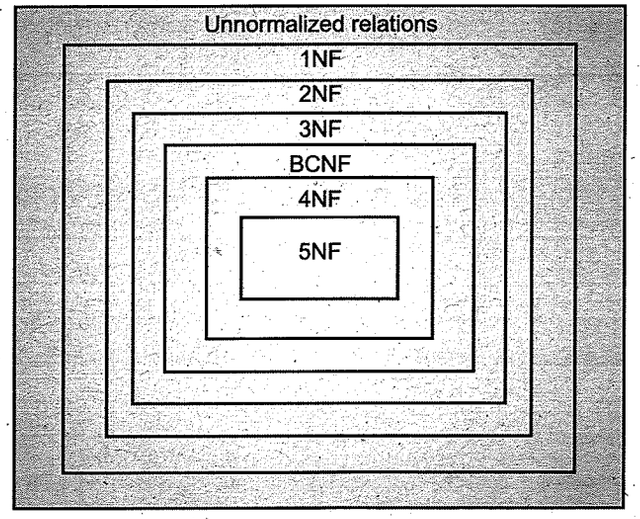


Fig. 2.50: Different Normal Forms

Purpose of Normalization:

- It is a technique for producing a set of relations with desirable properties given the data requirements of an enterprise.
- Normalization means to represent database design in a normal form.
- A normal form represents a good database design. It is used to eliminate various anomalies and inconsistencies.
- A relation that has any undesirable properties in the form of insertion, deletion or update anomalies, repetition of information or loss of information, is replaced by its projections. This results in a normal form.
- There are two approaches to represent the database design in a normal form:
  1. Decomposition.
  2. Synthesis.
- The decomposition approach starts with one relation and associated set of constraints in the form of:
  - Functional dependencies.
  - Multivalued dependencies.
  - Join dependencies.
- The second approach is synthesis approach. It starts with a set of functional dependencies on a set of attributes. It then synthesizes relations of Third Normal Form (3NF).

Issues of Relational Database Design:

1. Data Redundancy:

- The major aim of relational database design is to group attributes into relations to minimize data redundancy and thereby reduce the file storage space required by the implemented base relations.
- The problems associated with data redundancy are illustrated by comparing the staff, branch relations with StaffBranch relation.
- The relations have the form:

  Staff (Staff_No, S_Name, Position, Salary, Branch_No)

  Branch (Branch_No, B_address)

  StaffBranch (Staff_No,S_Name,Position,Salary,Branch_No,Address)

Table 2.13: Staff Table

Staff_No	S_name	Position	Salary	Branch_No
SL21	Rima Sinha	Manager	30000	B005
SG37	Raj Varma	Assistant	12000	B003
SG14	Varun S.	Supervisor	18000	B003

Contd...

SA9	Neeta K.	Assistant	9000	B007
SG5	Ram Shetty	Manger	24000	B003
SL41	Riya Khanna	Assistant	9000	B005

Table 2.14: Branch Table

Branch_No	B_Address
B005	22, MIDC Road, Baramati
B007	16, Law College Road, Pune
B003	34, 7 Street Road, Delhi

Table 2.15: StaffBranch

Staff_No	S_name	Position	Salary	Branch_No	Address
SL21	Rima Sinha	Manager	30000	b005	22, MIDC Road, Baramati
SG37	Raj Varma	Assistant	12000	B003	34, 7 Street Road, Delhi
SG14	Varun S.	Supervisor	18000	B003	34, 7 Street Road, Delhi
SA9	Neeta K.	Assistant	9000	B007	16, Law College Road, Pune
SG5	Ram Shetty	Manger	24000	B003	34, 7 Street Road, Delhi
SL41	Riya Khanna	Assistant	9000	B005	22, MIDC Road, Baramati

- In StaffBranch relation there is redundant data; the details of branch are repeated for every member. To avoid this problem we use normalization technique.

2. Anomalies:

- Relations that have redundant data may have problems called **update anomalies**, which are classified as insertions, deletions or modification anomalies.

(i) Insertion Anomalies: There are two main types of insertion anomalies, which can be illustrated using StaffBranch relation:

- To insert the details of new members of staff into 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 correct details of branch number B007, so that branch details are consistent with values for branch B007 in other tuples of the StaffBranch relation. The relations Staff and Branch do not suffer from this problem.
- 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

Staff_No. However Staff_No is the primary key for the StaffBranch relation. Attempting to enter null violates the entity integrity and is not allowed. We therefore can not enter a tuple for a new branch into the StaffBranch relation.

(ii) Deletion Anomalies: If we delete a tuple from the StaffBranch relation and that was the last member of the staff located at a branch, the details of the branch are also lost from the database.

For example: If we delete the tuple for staff number SA9, the details regarding B007 are lost from the StaffBranch relation.

(iii) Modification Anomalies: If we want to change the value of one of the attributes of a particular branch in the StaffBranch relation, we must update the tuples of all staffs at that branch.

For example: If we want to change the address for branch number B003, we must update the tuples for all staff located at that branch.

- One approach is to design relational schemas that are in appropriate normal forms. Normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database.

2.5.1 Types of Normal Forms

- There are four types of normal forms:

1. First Normal Form (1NF)

- A relation schema is said to be in First Normal Form (1NF) if the values in the domain of each attribute of the relation are atomic.
- In other words, only one value is associated with each attribute and the value is not a set of values or a list of values.
- A database schema is in First Normal Form (1NF) if every relation schema indicated in the database schema is in 1NF.

  Example: Course_prof relation shown in Table 2.16 is not in 1NF but shown in Table 2.17 is in 1NF.

Table 2.16: Course_Prof. relation

Fac_Dept	Prof.	Course preferences	
		Course	Course_Dept.
Comp-sci	Sachin	353	Comp sci
		379	Comp sci
		221	Decision sci
	Raj	353	Comp sci
		351	Comp sci
		379	Maths

Contd....

Chemistry	Rajesh	353	Comp sci
		456	Maths
		272	Chemistry
Mathematics	Rahul	353	Comp sci
		379	Comp sci
		221	Maths
		456	Maths

Table 2.17: Course_Prof. relation

Fac_Dept	Prof_name	Course	Course_Dept.
Comp sci	Sachin	353	Comp sci
Comp sci	Sachin	379	Comp sci
Comp sci	Sachin	321	Decision sci
Comp sci	Raj	353	Comp sci
Comp sci	Raj	351	Comp sci
Comp sci	Raj	379	Maths
Chemistry	Rajesh	353	Comp sci
Chemistry	Rajesh	456	Maths
Chemistry	Rajesh	272	Chemistry
Mathematics	Rahul	353	Comp sci
Mathematics	Rahul	379	Comp sci
Mathematics	Rahul	221	Maths
Mathematics	Rahul	456	Maths

- **Functional dependency**: A *Functional Dependency* (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below :

$$X \longrightarrow Y$$

- The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*.
- **Transitive dependency:** A transitional functional dependency can occur when changing a non-key column, might cause any of other non-key columns to change.

2. Second Normal Form (2NF)

- A relation schema R (S, F) is in Second Normal Form (2NF) if it is in the 1NF and if all non-key attributes are fully functionally dependent on the primary key (s).

- A database schema is in Second Normal Form (2NF) if every relation schema included in the database schema is in Second Normal Form.
- **Example:** Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Table 2.18: Teachers table

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non-prime attribute: teacher_age

- The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non-prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "*no non-prime attribute is dependent on the proper subset of any candidate key of the table*".
- To make the table complies with 2NF we can break it in two tables like this:

Table 2.19: teacher_details table

teacher_id	teacher_age
111	38
222	38
333	40

Table 2.20: teacher_subject table

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

- Now the tables comply with Second Normal Form (2NF).

3. Third Normal Form (3NF)

- In those cases where we cannot meet all three design criteria, we abandon BCNF and accept a weaker normal form called Third Normal Form (3NF).
- **Rule:** 3NF is applied when the relations are in 2NF and there is any non-key attribute that depends transitivity on the primary key.
- A decomposition in 3NF is always:
  - Loss-less join decomposition.
  - Dependency preserving decomposition.
- BCNF requires that all non-trival dependencies be of the form where a super key is. 3NF relaxes this constraint slightly by allowing non-trival functional dependencies whose left side is not a super key.
- **Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

Table 2.21: employee_details

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	Jaya	282005	UP	Agra	Dayal Bagh
1002	Andrew	222008	US	Washington	Columbia
1006	Lara	282007	TN	Chennai	Urrapakkam
1101	Raghav	292008	MH	Pimpri	Pune
1201	Sandesh	222999	MP	Gwalior	Ratan

  - **Super keys**: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}...so
  - **Candidate Keys**: {emp_id}
  - **Non-prime attributes**: All attributes except emp_id are non-prime as they are not part of any candidate keys.
- Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
- To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

Table 2.22: employee table

emp_id	emp_name	emp_zip
1001	Jaya	282005
1002	Andrew	222008
1006	Lara	282007
1101	Raghav	292008
1201	Sandesh	222999

Table 2.23: employee_zip table

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	US	Washington	Columbia
282007	TN	Chennai	Urrapakkam
292008	MH	Pimpri	Pune
222999	MP	Gwalior	Ratan

4. Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF, if every attribute on which some other attribute is fully functionally dependent is also a candidate for primary key of the relation.
- It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X→Y, X should be the super key of the table.
- **Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

Table 2.24(a): employee_details

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Indian	Production and Planning	D001	200
1001	Indian	Stores	D001	250
1002	American	Design and Technical Support	D134	100
1002	American	Purchasing Department	D134	600

Functional dependencies in the table above:
emp_id → emp_nationality
emp_dept → {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

- The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
- To make the table comply with BCNF we can break the table in three tables like this:

Table 2.24(b): emp_nationality table

emp_id	emp_nationality
1001	Indian
1002	American

Table 2.24(c): emp_dept table:

emp_dept	dept_type	dept_no_of_emp
production and planning	D001	200
stores	D001	250
design and technical support	D134	100
purchasing	D134	600

Table 2.24(d): emp_dept_mapping table

emp_id	emp_dept
1001	production and planning
1001	stores
1002	design and technical support
1002	purchasing

Functional dependencies:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

- This is now in BCNF as in both the functional dependencies left side part is a key.

Case Studies

Case 1: Normalize the following table upto 3 NF step by step. The database is related to order-report shown in following table.

Table 2.25: Consolidated Order Report

Order_no	Order_date	Invoice_no	Invoice_date	Product_no	Product_name	Quantity_ordered
25	17/3/13	1231	30/4/13	1	Computer	5
130	17/3/13	1232	30/4/13	2	Printer	5
520	28/3/13	1233	30/4/13	2	Printer	1
				4	A4 pages	10
				5	Main sheets	70
				6	Mouse	10

Solution:

- The given data is in unnormalized form.
- First convert it into 1 NF.

1 NF:

- No repeating groups.
- For this, partition each data structure containing repeating groups which accomplishes the same purpose.

Table 2.26(a): Table with repeating groups

Order_no	Order_date	Invoice_no	Invoice_date	Product_no	Product_name	Quantity_ordered
25	17/3/13	1231	30/4/13	1	Computer	5
130	17/3/13	1232	30/4/13	2	Printer	5
520	28/3/13	1233	30/4/13	2	Printer	1
520	28/3/13	1233	30/4/13	4	A4 pages	10
520	28/3/13	1233	30/4/13	5	Main sheets	70
520	38/3/13	1233	30/4/13	6	Mouse	10

- We remove the four repeating elements and group them into a new table called **order_record**. [See table 2.26 (b)].
- Rest of the elements are added into a new table called **order_item**. [See table 2.26 (c)]

Table 2.26 (b): Order_record

Order_no	Order_date	Invoice_no	Invoice_date
25	13/3/13	1231	30/4/13
130	17/3/13	1232	30/4/13
520	28/3/13	1233	30/4/13

Table 2.26 (c): Order_item

Order_no	Product_no	Product_name	quantity
25	1	computer	5
130	2	printer	5
520	2	printer	1
	4	A4 pages	10
	5	mainsheets	70
	6	mouse	10

- Both the relations order_record and order_item are in 1 NF.
- But order_item is not in its ideal form.
- Some of the attributes are not functionally dependent on primary key (order_no, product_no).
- Some non-key attributes such as product name, quantity are dependent only on product_no and not on the key which we have considered.

2 NF:

- All the non-key fields are dependent on the primary key (composite key).
- For this, verify that each non-key field is in first normal form and fully functionally dependent on primary key.
- Product no → product description is the functional dependency.
- To convert order_item into 2 NF, we separate the table (decompose) such that the above functional dependency satisfies correctly. [See tables 2.27 (a), 2.27 (b), 2.27 (c)]

Table 2.27 (a): Order_record

Order_no	Order_date	Invoice_no	Invoice_date
25	13/3/13	1231	30/4/13
130	17/3/13	1232	30/4/13
520	28/3/13	1233	30/4/13

Table 2.27 (b): Order_item

Order_no	Product_no	Quantity_ordered
24	1	5
130	2	5
520	2	1
	4	10
	5	70
	6	10

Table 2.27 (c): Product

Product_no	Product_ description
1	computer
2	printer
4	A4 pages
5	mainsheets
6	mouse

3 NF:

- When all non-key fields are independent of one another, then relation is in 3 NF.
- There is check for redundancy within the relation.
- Duplicate data elements which can be derived from other elements are removed at 3 NF.

- For example, in order-record relation we have invoice date which is dependent on both i.e. order_no and invoice_no.
- To convert the Order_Record into 3 NF we have to remove invoice_date and group it with invoice_no as the key in separate table.
- So we have following 3 NF relations: [See Tables 2.28 (a), (b), (c), (d)]

Table 2.28 (a): Order-record

Order_no	Order_date	Invoice_no
25	13/3/13	1231
130	17/3/13	1232
520	28/3/13	1233

Table 2.28 (b) Order-item

Order_no	Product_no	Quantity_ ordered
25	1	5
130	2	5
520	2	1
520	4	10
520	5	70
520	6	10

Table 2.28 (c): Product

Product_no	Product_description
1	computer
2	printer
4	A4 pages
5	mainsheets
6	mouse

Table 2.28 (d): Invoice

Invoice_no	Invoice_date
1231	30/4/13
1232	30/4/13
1233	30/4/13

Therefore the relationship is transformed into following relations (3 NF).

- **Order_record** – Order_no – primary key
  Order_date
  Invoice_no – foreign key
- **Product** – Product_no – primary key
  Product_ description
- **Order_item** – Order_no – primary key
  Product_no – foreign key
  Quantity_ordered
- **Invoice** – Invoice_ no – primary key
  Invoice_date

Case 2: The given Sales_report in un-normalized relation since it has repeating groups. Normalize this relation upto 3 NF. The Sales_report is shown in table 2.29

Table 2.29: Sales_report

Sales_ person_ no	Sales_ person_ name	Sales_ area	Cust_no	Cust_name	Warehouse_ no	Warehouse_ location	Sales_ amount
501	Vinit	Aundh	101 102 301	CMS IBM Lavasa	4 3 4	Mumbai Thane Mumbai	2 lakh 2.5 lakh 4 lakh
345	Pritesh	Warje	45 127	Infosys Microland	2 1	Chennai Bangalore	55 Thousand 1.2 lakh

Solution: The normalized sales report is as shown in Table. 2.30.

Table 2.30: Normalized Sales Report

Sales_ person_ no	Sales_ person_ name	Sales_ area	Cust_no	Cust_name	Warehouse_ no	Warehouse_ location	Sales_ amount
501	Vinit	Aundh	101	CMS	4	Mumbai	2 lakh
501	Vinit	Aundh	102	IBM	3	Thane	2.5 lakh
501	Vinit	Aundh	301	Lavasa	4	Mumbai	4 lakh
345	Pritesh	Warje	45	Infosys	2	Chennai	55 thousand
345	Pritesh	Warje	127	Microland	1	Banglore	1.2 lakh

1 NF:

- Create two separate structures Sales_person (Table. 2.31(a)) and Sales_customer (Fig. 2.31(b)).

Table. 2.31 (a): Sales_person

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Table. 2.31 (b): Sales_Customer

Sales_person_no	cust_no	cust_name	warehouse_no	w_location	Sales_amount
501	101	CMS	4	Mumbai	2 lakh
501	102	IBM	3	Thane	2.5 lakh
501	301	Lavasa	4	Mumbai	4 lakh
345	45	Infosys	2	Chennai	55 thousand
345	127	Microland	1	Banglore	1.2 lakh

- In sales_customer, some of the attributes are not functionally dependent on primary key_sales person no, cust_no.

2 NF:

- Sales_person table is as it is. [See Table 2.32 (a)]
- Sales_customer will be decomposed into two more relations i.e. sales [See Table 2.32 (b) and warehouse [See Table 2.32 (c)].

Table 2.32 (a): Sales_person

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Table 2.32 (b): Sales

Sales_person_no	Cust_no	Sales_Amount
501	101	2 lakh
501	102	2.5 lakh
501	301	4 lakh
345	45	55 thousand
345	127	1.2 lakh

Table 2.32 (c): Warehouse

cust_no	cust_name	warehouse_no	warehouse_location
101	CMS	4	Mumbai
102	IBM	3	Thane
301	Lavasa	4	Mumbai
45	Infosys	2	Chennai
127	Microland	1	Bangalore

- Warehouse location (warehouse location) is not fully dependant on cust_no i.e. it has non-key dependencies.
- We will go for 3 NF.

3 NF:

- Sales person and sales relations are as it is. [See Table 2.33 (a), Table 2.33(b)].
- Warehouse is decomposed into customer [Table 2.33 (c)] and warehouse [Table 2.33(d)].

Fig. 2.33 (a): Sales_person

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Fig. 2.33 (b): Sales

Sales_person_no	Cust_no	Sales_amount
501	101	2 lakh
501	101	2.5 lakh
501	301	4 lakh
345	45	55 thousand
345	127	1.5 lakh

Fig. 2.33 (c): Customer

Cust_no	Cust_name	Warehouse_no
25	CMS	4
102	IBM	3
301	Lavasa	4
45	Infosys	2
127	Microland	1

Fig. 2.33 (d): Warehouse

warehouse_no	warehouse_location
4	Mumbai
3	Thane
2	Chennai
1	Bangalore

- Therefore, the sales_report relation is transformed into following 3NF relations:
- **Sales_person** – Sales_person_no – primary key
  Sales_person_name
  Sales_area
- **Sales** – Cust_no – foreign key
  Sales_amount
  Sales_person_no – foreign key
- **Customer** – Cust_no – primary key
  Cust_name
  Warehouse_no – foreign key
- **Warehouse** – Warehouse_no – primary key
  Warehouse_location

Case 3: Normalize the following data.

project_no
project_name
emp_no
emp_name
rate_category
hourly_rate

Solution: Above is the unnormalized data.

1 NF: Separate repeating groups

Employee-project table:

project_no – primary key
project_name
emp_no – primary key
emp_name
rate_category
hourly_rate

2 NF: Find the elements depending upon primary keys or partial keys.

Employee Project Table:

project_no – primary key
emp_no – primary key

Employee table:

emp_no – primary key
emp_name
rate_category
hourly_rate

Project table:

project_no – primary key
project_name

3 NF: Find non-key attributes. Emp_name is not dependent on either rate_category or hourly_rate. Same is applied to rate_category. But, hourly_rate is dependent on rate_ category.

Employee project table:

project_no – primary key
emp_no – primary key

Employee table:

emp_no – primary key
emp_name
rate_category

Rate table:

rate_category – primary key
hourly_rate

Project table:

project_no – primary key
project_name

All above tables are in 3 NF and ready to implement.

Case 4: Normalize the following table of attributes up to 3 NF. The database is related to cricket association.

- The fields are:

Team_code	skill_name	club_city
player_code	club_code	club_district
team_name	umpire_code	
player_name	club_name	
player_age	umpire_name	
player_skill_code	club_address	

- Following assumptions to be made –
  1. A player can belong to one team and has only one skill.
  2. An umpire can belong to one club only.
  3. Club arranges matches between various teams and appoint umpires for the matches.

Solution: Unnormalized data:

club_code	team_code
club_name	team_name
club_address	player_code
club_district	player_name
club_city	player_age
umpire_code	player_skill_code
umpire_name	skill_name

1NF: Separate out the repeating group and find out key field.

Club table

club_code – primary key
team_code – secondary key
team_name
player_code
player_name
player_age
player_skill_code
skill_name

Umpire table

club_code – primary key
club_name
club_address
club_district
club_city
umpire_code
umpire_name

2NF: Find out the attributes depending upon the whole key and also the attributes depending upon the partial key.

CT table

club_code – primary key
team_code – secondary (partial key)

Team table

team_code – primary key
team_name

Player table

team_code
player_code – primary key
player_name
player_age
player_skill_code
skill_name

Club table

club_code – primary key
club_name
club_address
club_district
club_city

Umpire table

club_code
umpire_code – primary key
umpire_name

3 NF: Find out the non-key elements depending upon some other non-key element. Extract the elements which can be directly computed.

From the Player table player_age, player_skill_code, skill_name can be separated out as they are non key elements depending upon the non key element namely player_name.

Similarly, club_name is a non key element.

Player table

player_name – primary key
player_age
player_skill_code
skill_name

Club table

club_name – primary key
club_address
club_district
club_city

The key elements from the new structures are found out. In the Player table player_name is set out as a key field. In the club table, club_name is set out as a key field.

Summary

- Data Model is the modelling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction.
- The Relational database model represents the database as a collection of relations (tables).
- Attribute, Tables, Tuple, Relation Schema, Degree, Cardinality, Column, Relation instance, are some important components of Relational Model.

- Relational Integrity constraints are referred to conditions which must be present for a valid relation.
- Domain constraints can be activated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.
- Insert, Select, Modify and Delete are operations performed in Relational Model.
- The relational database is only concerned with data and not with a structure which can improve the performance of the model.
- The full form of E-R is Entity and Relationships Diagrams. They play a very important role in the database designing process.
- An Entity–relationship model (E-R model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (E-R Diagram).
- E-R diagram has three main components: Entity, Attribute, and Relationship.
- An entity is an object or component of data. An attribute describes the property of an entity. A relationship is used to describe the relation between entities.
- There are four types of relationships: One to One, One to Many, Many to One, Many to Many.
- Normalization is a process of refining database structures to improve the speed at which data can be accessed and to increase database integrity.
- The most commonly used normal forms: First normal form(1NF), Second normal form(2NF), Third normal form(3NF), Boyce & Codd normal form (BCNF).
- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- Specialization and Aggregation are the ways to represent special relationships between entities and attributes in E-R Model in DBMS.
- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

Check Your Understanding

1. A relational database developer refers to a record as _____.
   (a) a criteria (b) a relation
   (c) a tuple (d) an attribute
2. A _____ is a set of column that identifies every row in a table.
   (a) composite key (b) candidate key
   (c) foreign key (d) super key

3. A relation is in _____, if an attribute of a composite key is dependent on an attribute of another composite key.
   (a) 2NF (b) 3NF
   (c) BCNF (d) 1NF
4. Data model is a collection of conceptual tools for describing _____.
   (a) Data (b) Data schema
   (c) Consistency constraints (d) All of these
5. A primary key should be defined as _____.
   (a) NULL (b) NOT NULL
   (c) Either of the above can be used (d) None of the above are correct
6. In the _____ normal form, a composite attribute is converted to individual attributes.
   (a) First (b) Second
   (c) Third (d) Fourth
7. Tables in Second Normal Form (2NF):
   (a) Eliminate all hidden dependencies.
   (b) Eliminate the possibility of insertion anomalies.
   (c) Have a composite key.
   (d) Have all non-key fields depend on the whole primary key.
8. Which of the following is a top-down approach in which the entity's higher level can be divided into two lower sub-entities?
   (a) Aggregation (b) Generalization
   (c) Specialization (d) All of the above
9. The entity relationship set is represented in E-R diagram as_____.
   (a) Double diamonds (b) Undivided rectangles
   (c) Dashed lines (d) Diamond
10. Weak entity set is represented as _____.
   (a) Underline (b) Double line
   (c) Double diamond (d) Double rectangle

ANSWER KEY

1. (c)	2. (d)	3. (b)	4. (d)	5. (b)
6. (a)	7. (a)	8. (c)	9. (d)	10. (c)

Practice Questions

Q.I: Answers the following questions in short.

1. Define entity with examples.

2. Explain symbols used in E-R diagram.
3. What is relationship and relationship set?
4. Describe the structure of relational database.
5. Explain the mapping cardinality of Database.

Q.II: Answers the following questions.

1. Explain E.F. Codd's rules in detail.
2. Explain generalization, aggregation with examples.
3. What is normalization? Explain types of normal forms.
4. Draw an ERD and normalize the following case up to 3 NF.

   Organization made up of various departments, each having a name, identifying no and an employee who is the manager. A department may be located in different places. Information about employee includes name, identification number, birth data, address, gender and salary. Each employee is assigned to one department. The data the manager is appointed to a department is also tracked. Employees may be directly supervised by another employee. Each project with in the organization is controlled by a department employees (not necessarily from the controlling dept.) are assigned to projects. Information about projects includes project name, no and location. Hours spent by employees on each project are also kept.
5. Design the E-R model for the following case and normalized it upto 3 NF.

   XYZ hospital is a multi-specialty hospital that includes a number of departments, rooms, doctors, nurses, compounders and other staff working in the hospital. Patients having different kinds of diseases come to the hospital and gets check-up done from the concerned doctors. If required they are admitted in the hospital and discharged after treatment.

   The aim of this case study is to design and develop a database for the hospital to maintain records of various departments, rooms and doctors in the hospital. It also maintains records of the regular patients; patients admitted in the hospital; the check-up of patients done by the doctors; the patients that have been operated and patient discharged from the hospital.

Q.III: Write a short note on:

1. Keys in DBMS
2. Specialization process
3. Constraints
4. Weak and strong entity
5. Types of Attribute

❖❖❖

3... Transaction and Concurrency Control

Objectives...

- To understand concept of transaction processing and serializibility.
- To know about Concurrency control and schemes such as locking techniques, timestamp based protocols.
- To learn about granularity of data items, deadlocks.

3.1 INTRODUCTION

- Collection of operations that forms a single logical unit of work is called **transaction.**
- Usually, a transaction is the result of execution of a user program, written in a high-level data manipulation language or programming language.
- Every transaction is delimited by statements or function calls of the form begin transaction and end transaction.
- The reliability of DBMS is linked to the reliability of computer system, and some solution must be there to deal with such computer system failures.
- Recovery system, the main component of Transaction management/Processing unit, deals with such failures. It makes the database fault tolerant.
- Number of transactions can be executed at the same time and they may be accessing the same database. Such concurrent access to the database may result in some inconsistent state of database.
- Concurrency control unit of transaction management preserves the consistency of database in case of concurrent accesses.

3.2 CONCEPT OF TRANSACTION PROCESSING

- A transaction is a program unit whose execution accesses and possibly updates the contents of a database.
- Transaction processing means dividing information processing up into individual, indivisible operations, called transactions.

- Transaction processing is designed to maintain database integrity (the consistency of related data items) in a known, consistent state.
- A transaction, a typical example of which would be a customer order, consists of a series of events (accepting the order, allocating stock and so forth) that are treated as a whole.

Types of Transactions:

1. **Based on Application:**
   - Non-distributed vs. distributed
   - Compensating transactions
   - Transactions Timing
   - On-line vs. batch
2. **Based on Actions:**
   - Two-step
   - Restricted
   - Action model
3. **Based on Structure:**
   - Flat or simple transactions: It consists of a sequence of primitive operations executed between begin and end operations.
   - Nested transactions: A transaction that contains other transactions.
   - Workflow.

3.2.1 ACID Properties

- To ensure the integrity of data, database system maintains following properties of transaction.

1. **Atomicity:** Atomicity property ensures that at the end of the transaction, either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of a transaction, the updates made by the transaction will be accessible to other transactions and processes outside the transaction.
2. **Consistency:** Consistency property of transaction implies that if the database was in consistent state before the start of a transaction, then on termination of a transaction, the database will also be in a consistent state.
3. **Isolation:** Isolation property of transaction indicates that action performed by a transaction will be hidden from outside the transaction until the transaction terminates. Thus each transaction is unaware of other transactions executing concurrently in the system.
4. **Durability:** Durability property of a transaction ensures that once a transaction completes successfully (commits), the changes it has made to the database persist, even if there are system failures.

- These four properties are often called ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability) properties of transaction.

3.2.2 States of Transaction

- Following are the possible states of a transaction during its execution.

1. **Active:** Transaction is active when it is executing. This is the initial state of transaction.
2. **Partially committed:** When a transaction completes its final statement, it enters in partially committed state.
3. **Failed:** If the system decides that the normal execution of the transaction can no longer proceed, then transaction is termed as failed.
4. **Committed:** When the transaction completes its execution successfully it enters committed state from partially committed state.
5. **Aborted:** To ensure the atomicity property, changes made by failed transaction are undone i.e. the transaction is rolled back. After rollback, that transaction enters in aborted state.

- A state is said to be terminated if it is committed or aborted.

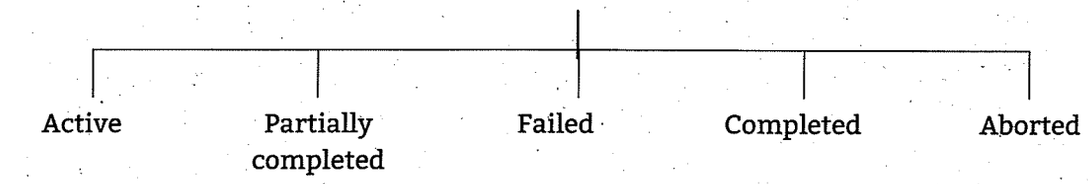


Fig. 3.1: Types of Transaction States

- The state diagram corresponding to transaction states is given in Fig 3.2.

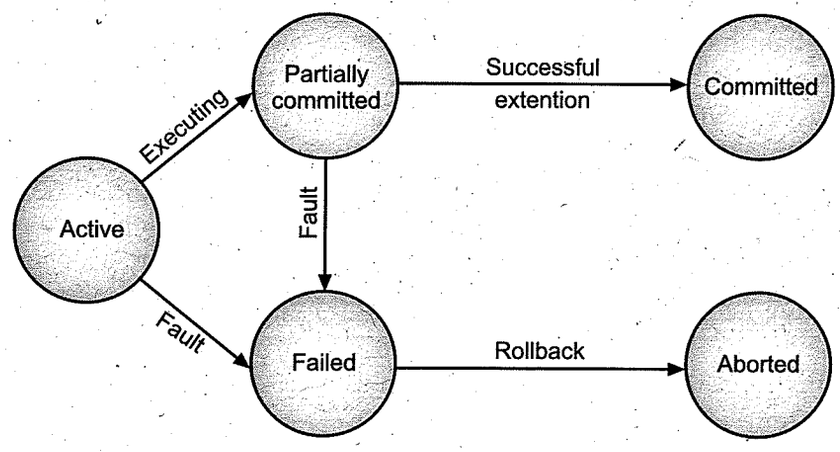


Fig. 3.2: State diagram of Transaction

- A transaction always starts with Active state. It remains in active state till all commands of that transaction are executed.

- When it completes executing the last command of that transaction, it enters Partially Committed state and when the transaction execution is successfully completed, it enters Committed state.
- If some failure occurs in Active state or Partially Committed state, transaction enters Failed state. When the transaction is in failed state, it rolls backs that transaction enters aborted state.
- When the transaction is in aborted state, the system has two options:
  1. If the transaction was aborted as a result of some hardware or software error (software error which is not created because of some internal logic of transaction), then such transaction can be restarted. A restarted transaction is considered to be a new transaction.
  2. If the transaction was aborted because of some internal logical error which can be corrected only by rewriting of the application program or because the input was bad or because the desired data were not found in the database, then system can kill such transactions.

3.3 CONCURRENCY CONTROL, PROBLEMS IN CONCURRENCY CONTROLS

3.3.1 Concurrency Control

- In a database management system (DBMS), concurrency control manages simultaneous access to a database. It prevents two users from editing the same record at the same time and also serializes transactions for backup and recovery.
- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. These problems are commonly referred to as concurrency problems in database environment.
- Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems. The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

3.3.2 Problems in Concurrency Controls

- The five concurrency problems that can occur in database are:

1. **Temporary Update Problem/dirty read problem:**

- Temporary update or dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

Example:

Table 3.1: Dirty Read Problem

T_1	T_2
read_item(X) X = X – N write_item (X)	
	read_item(X) X = X + M Write_item(X)
read_item(Y)	

- In the above example, if T_1 fails for some reason then X will revert back to its previous value. But T_2 has already read the incorrect value of X.

2. Incorrect Summary Problem:

- Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.

Example:

Table 3.2: Incorrect Summary Problem

T_1	T_2
	sum = 0 read_item(A) sum = sum + A
read_item(X) X = X – N Write_item(X)	
	read_item(X) sum = sum + X read_item(Y) sum = sum + Y
read_item(Y) Y = Y + N write_item(Y)	

- In the above example, T_2 is calculating the sum of some records while T_1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.

3. Lost Update Problem:

- In the lost update problem, update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Example:

Table 3.3: Lost Update Problem

T_1	T_2
read_item(X) X = X + N	
	X = X + 10 Write_item(X)

- In the above example, T_1 changes the value of X but it gets overwritten by the update done by T_2 on X. Therefore, the update done by T_1 is lost.

4. Unrepeatable Read Problem:

- The unrepeatable read problem occurs when two or more read operations of the same transaction read different values of the same variable.

Example:

Table 3.4: Unrepeatable Read Problem

T_1	T_2
Read(X)	
	Read(X)
Write(X)	
	Read(X)

In the above example, once T_2 reads the variable X, a write operation in T_1 changes the value of the variable X. Thus, when another read operation is performed by T_2, it reads the new value of X which was updated by T_1.

5. Phantom Read Problem:

- The Phantom Read Problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

Example:

Table 3.5: Phantom Read Problem

T_1	T_2
Read(X)	
	Read(X)

Contd...

Delete(X)	
	Read(X)

- In the above example, once T_2 reads the variable X, T_1 deletes the variable X without T_1's knowledge. Thus, when T_2 tries to read X, it is not able to do it.

3.4 SCHEDULING OF TRANSACTIONS, SERIALIZABILITY AND TESTING OF SERIALIZABILITY

- In this section, we will understand concept of schedule and serializability. Then we will see how to schedule transactions and test serializability.

3.4.1 Schedule

- Schedule represents the sequential order in which instructions are executed in the system.
- Two schedule types are Serial Schedule and Concurrent Schedule.
  1. **Serial Schedule:** Serial schedule is a type of schedule where no transaction ever interleaved with each other i.e. after complete execution of a transaction; another transaction begins its execution. It consists of a sequence of instructions from various transactions where the instructions belonging to one single transaction appear together in that schedule. Thus for a set of n transactions, there exist n! different valid serial schedules.
  2. **Concurrent Schedule:** When several transactions are executed concurrently, the corresponding schedule is called Concurrent Schedule. Several execution sequences are possible since the various instructions from both transactions may now be interleaved.

     In general, it is not possible to predict exactly how many instructions of a transaction will be executed before CPU switches to other transaction. Thus the number of possible schedules for a set of n transactions is much larger than n!
- **Examples of Schedules:** Consider the banking system of several accounts and a set of transactions that accesses and updates those accounts. Let T_1 and T_2 be two transactions.

  T_1: Transfers ₹ 50 from account A to account B.

  T_1 : read (A)
  A := A – 50
  write (A)
  read (B)
  B := B + 50
  write (B)

T_2: Transfers 10% of balance from account A to B.

T_2 : read (A)
temp: = A * 0.1
A:= A – temp
write (A)
read (B)
B:= B + temp
write (B)

For T_1 and T_2, two serial transactions are possible:

1. **<T_1, T_2> Serial Schedule:** It will execute transaction T_1 first and then T_2.
   After executing both transactions, account balance of A is ₹ 855 and account balance of B is ₹ 2145 i.e. <T_1 T_2> preserves A + B i.e. the database is consistent.

Table 3.6: Schedule 1

T_1	T_2
read (A)	
A:= A – 50	
write (A)	
read (B)	
B:= B + 50	
Write (B)	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
	read (B)
	B:= B + temp
	write (B)

2. **<T_2, T_1> Serial Schedule:** It executes T_2 first and then T_1.

Table 3.7: Schedule 2

T_1	T_2
	read (A)
	temp:= A * 0.1

Contd...

	A:= A – temp
	write (A)
	read (B)
	B:= B + temp
	write (B)
read (A)	
A:= A – 50	
write (A)	
read (B)	
B:= B + 50	
write (B)	

- Here, after executing both the transactions, values of A and B are ₹ 850 and ₹ 2150 respectively. In this case also A + B are constant and it preserves the consistency.
- For T_1 and T_2, number of concurrent schedules is possible. But, not all the transactions are consistency preserving.
- A concurrent schedule for T_1 and T_2 is given in Table 3.8.

3. Consistent concurrent schedule for T_1 and T_2:

This concurrent schedule preserves the consistency of database and A + B is constant.

Table 3.8: Schedule 3

T_1	T_2
read (A)	
A:= A – 50	
write (A)	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
read (B)	
B:= B + 50	
write (B)	
	read (B)
	B:= B + temp
	write (B)

4. Inconsistent concurrent schedule for T_1 and T_2:

Table 3.9: Schedule 4 (concurrent schedule)

T_1	T_2
read (A)	
A:= A – 50	
	read (A)
	temp:= A * 0.1
	A:= A – temp
	write (A)
	write (B)
write (A)	
read (B)	
B:= B + 50	
write (B)	
	B:= B + temp
	write (B)

3.4.2 Serializability

- The schedule that formed by swapping non-conflicting instructions are called as Serializability and the schedule formed is called **Serializable Schedule**.
- For a transaction, always a serial schedule results in a consistent database and not every concurrent schedule can result in consistent database. But a concurrent schedule results in a consistent state if its result is equivalent to a serial schedule of that transaction. Such concurrent schedule is known as **serializable**.
- Serializability theory in DBMS deals with maintaining the correctness of a concurrent execution. When two transactions are executed concurrently, there is no way to check whether any problem has been occurred or not.
- A **Serializable Schedule** is defined as: Given (an interleaved execution) a concurrent schedule for n transactions; the following conditions hold for each transaction in the set.
  1. All transactions are correct i.e. if any one of the transactions is executed on a consistent database, the resulting database is also consistent.
  2. Any serial execution of the transactions is also correct and preserves the consistency of the database.
- The given concurrent schedule is said to be serializable if it produces the same result as some serial schedule of the transaction.

- There are two forms of serializability:
  1. Conflict serializability
  2. View serializability

3.4.2.1 Conflict Serializability

- The Rule of conflict serializability status that a schedule can be conflict serializable if the following conditions hold:
  1. The two operations **(Read and write)** in a schedule belong to different transactions.
  2. The two operations access the same data item.
  3. One of the two operations is a **write** operation.
- Consider that T_1 and T_2 are two transactions and S is a schedule for T_1 and T_2. I_i and I_j are two instructions. If I_i and I_j refer to different data items, then I_i and I_j can be executed in any sequence.
- But, if I_i and I_j refer to same data items then the order of two instructions may matter. Here, I_i and I_j can be a read or write operation only. Hence, following conditions are possible:

  Case 1: I_i = read (A)

  I_j = read (A)
- The order of I_i and I_j does not matter because both are reading the data.

  Case 2: I_i = read (A) I_j = write (A)

  I_i = write (A) I_j = read (A)
- Here, if read (A) is executed before write (A) then it will read the original value of A, otherwise it will read that value of A which is written by write (A). Hence, the order of I_i and I_j matters.

  Case 3: I_i = write (A) I_j = write (A)
- Here, order of I_i and I_j does not affect either T_i or T_j. But the database is changed and it makes difference for next read.
- We say that I_i and I_j conflict if they are operated by different transactions on the same data item and at least one of them is write operation i.e. only in Case 1, I_i and I_j do not conflict.
- Consider an example of concurrent schedule 5.

Table 3.10: Concurrent schedule 5

T_1	T_2
read (A)	
write (A)	
	read (A)

Contd...

	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

- Here, write (A) of T_1 conflicts with read (A) of T_2; similarly write (B) of T_1 conflicts with read (B) of T_2. But write (A) of T_2 does not conflict with read (B) of T_1 because both are accessing different data items. If I_i and I_j are two consecutive instructions of schedule S and if they do not conflict, then we can swap the order of I_i and I_j, to produce new schedule S'. We say that S and S' are equivalent since all instructions appear in the same order except for I_i and I_j whose order does not matter.
- Equivalent schedule for a schedule given in Table 3.10 can be obtained by following swap. Swap write (A) of T_2 with read (B) of T_1.

Table 3.11: Schedule 6 (Schedule after swapping instructions)

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	
	read (B)
	write (B)

- Similarly, swap the following in schedule given in Table 3.11.
  1. read (B) instruction and read (A) instruction of T_1 and T_2 respectively.
  2. write (B) instruction and write (A) instruction of T_1 and T_2 respectively.
  3. write (B) instruction and read (A) instruction of T_1 and T_2 respectively.
- The final schedule S' after these swapping is given below:

Table 3.12: Schedule 7

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

- This is a serial schedule of T_1 and T_2. Thus, concurrent schedule S is transferred to serial schedule S' by a series of swaps of non-conflicting instructions and schedules S and S' are conflict equivalent. We say that a schedule S is conflict serializable, if it is conflict equivalent to a serial schedule.
- Consider the schedule shown in Table 3.13.

Table 3.13: Schedule 8

T_3	T_4
read (Q)	
	write (Q)
read (Q)	

- This schedule is not conflict serializable, since it is not conflict equivalent to any serial schedule $<T_3, T_4>$ or $<T_4, T_3>$.
- There may be any two schedules which are not conflict equivalent but produce same outcome. Schedule given in Table 3.14 is not conflict serializable.

Table 3.14: Schedule 9 (Concurrent schedule)

T_1	T_5
read (A) A:= A – 50 write (A)	
	read (B) B:= B – 10 write (B)
read (B) B:= B + 50 write (B)	
	read (A) A:= A + 10 write (A)

- Result of above schedule is same as serial schedule $<T_1, T_5>$, but this is not conflict serializable, since, in above schedule write (B) of T_5 conflicts with read (B) of T_1. Thus, we cannot move all instructions of T_1 before those of T_5 by swapping consecutive non-conflicting instructions.

3.4.2.2 View Serialzibility

- Consider two schedules S and S', where same set of transactions participate in both schedules. The schedules S and S' are said to be view equivalent, if the following three conditions are satisfied:

1. For each data item Q if transaction T_i reads the initial value of Q in schedule S, then transaction T_i in schedule S' must also read the initial value of Q.
2. For each data item Q if transaction T_i executes read (Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i in schedule S', must also read the value of Q that was produced by T_j transaction.
3. For each data item Q, the transaction that performs the final write (Q) operation in schedule S must perform the final write (Q) operation in schedule S'.

- A schedule S is view serializable if it's view equivalent to a serial schedule.
- **Example of view equivalence:** Schedule 1 is not view equivalent to Schedule 2 since, in Schedule 1 the value of account A read by transaction T_2 was produced by T_1, whereas this is not the case in Schedule 2. Schedule 1 is view equivalent to Schedule 3, because values of account A and B read by transaction T_2 were produced by T_1 in both schedules.
- **Example of view serializable schedule:** A schedule given in Table 3.15 is view serializable schedule.

Table 3.15: Schedule 10

T_3	T_4	T_6
read (A)		
	write (A)	
write (A)		
		write (A)

- This schedule is view equivalent to serial schedule $<T_3, T_4, T_6>$.

  Note: Transactions T_4 and T_6 perform write(A) operations without having performed a read (A) operation. Writes of this form are called *blind writes*.
- Every conflict serializable schedule is view serializable, but there are view serializable schedules that are not conflict serializable.
- A view serializable schedule in which blind write appear is not a conflict serializable. Schedule 10 is view serializable but it is not conflict serializable.

3.4.3 Testing for Serialzibility

- A serializability schedule gives same result as some serial schedule. A serial schedule always gives correct result, i.e. a schedule that is serializable schedule is always correct.
- Hence, we must show that schedules generated by concurrency control scheme are serializable. This section deals with methods for determining conflict and view serializability.

1. Conflict serializability:

There is an algorithm to establish the serializability of a given schedule for a set of transactions. This algorithm uses a Directed Graph called Precedence Graph, constructed from given schedule.

- **Precedence Graph:** It consists of a pair G = (V, E)

  where,

  V – Set of vertices.

  The set of vertices consists of all transactions participating in the schedule.

  E – Set of edges.

  The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of the following three conditions hold:

  (i) T_i executes write (Q) before: T_j executes read (Q)

  (ii) T_i executes read (Q) before: T_j executes write (Q)

  (iii) T_i executes write (Q) before: T_j executes write (Q)

- A precedence graph is said to be acyclic if there are no cycles in the graph otherwise it is a Cyclic graph.

Algorithm: Conflict Serializability:

Step 1 : Construct a precedence graph G for given schedule S.

Step 2 : If the graph G has a cycle, schedule S is not conflict serializable.

If the graph is acyclic, then find, using the topological sort given below, a linear ordering of transactions, so that if there is arc from T_i to T_j in G, T_i precedes T_j. Find a serial schedule as follows:

(i) Initialize the serial schedule as empty.

(ii) Find a transaction T_i, such that there are no arcs entering T_i, T_i is the next transaction in the serial schedule.

(iii) Remove T_i and all edges emitting from T_i. If the remaining set is non-empty, return to (ii), otherwise the serial schedule is complete.

Examples:

(i) Given schedule is:

Table 3.16: Schedule 11

T_{11}	T_{12}	T_{13}
read (A)		
	read (B)	
A:= f_1 (A)		
		read (C)
	B:= f_2 (B)	
	write (B)	

Contd...

		C:= f_3 (C)
		write (C)
write (A)		
	read (A)	
	A:= f_4 (A)	
read (C)		
	write (A)	
C:= f_5 (C)		
write (A)		
		B:= f_6 (B)
		write (B)

- Precedence graph for the schedule is shown in Fig. 3.3.

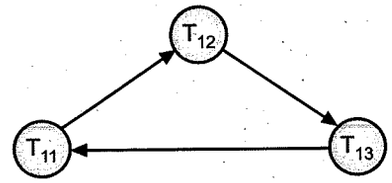


Fig. 3.3: Precedence Graph

- This graph contains cycle. Hence, the schedule is not conflict serializable.

(ii) The given schedule is:

Table 3.17: Schedule 12

T_{14}	T_{15}	T_{16}
read (A)		
A:= f_1 (A)		
read (C)		
write (A)		
A:= f_2 (C)		
	read (B)	
write (C)		
	read (A)	
		read (C)
	B:= f_3 (B)	
	write (B)	
		C:= f_4 (C)

		read (B)
		write (C)
	A:= f_5 (A)	
	write (A)	
		B:= f_6 (B)
		write (B)

- The precedence graph for the given schedule is shown in Fig. 3.4.

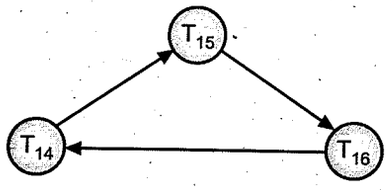


Fig. 3.4: The Precedence Graph

- The graph is acyclic. The conflict equivalent serial schedule for given schedule can be obtained using step 2 of algorithm. T_{14} is the transaction with no arcs entering in T_{14}. Hence, T_{14} is the first transaction in serial schedule. Remove T_{14} and all edges emitting from T_{14}. T_{15} is the next schedule, since it has no incoming edges. Remove T_{15} and edges emitting from T_{15}. T_{16} is the last schedule. Hence the serial schedule which is conflict equivalent to given schedule is shown in Fig. 3.5.

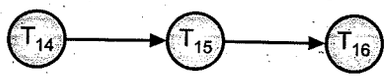


Fig. 3.5: Serial Schedule

Hence, the schedule is conflict serializable.

(iii) Consider schedule 3, the precedence graph for it is given as:

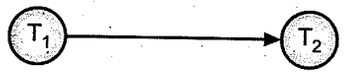


Fig. 3.6

Since, T_1 executes write (A) and write (B) before T_2 executes read (A) and read (B).

This graph is acyclic and the conflict equivalent serial schedule is $T_1 \rightarrow T_2$. Hence, schedule 3 is conflict serializable.

(iv) Consider schedule 4, the precedence graph for it is:

Fig. 3.7

This is a cyclic graph and hence it is not conflict serializable.

(v) Consider the precedence graph given in Fig. 3.8.

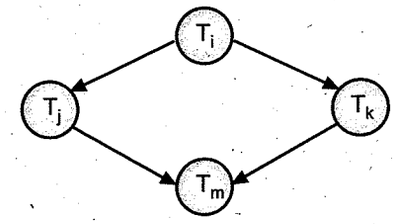


Fig. 3.8

The graph is acyclic. The conflict equivalent serial schedule is equivalent to this are:

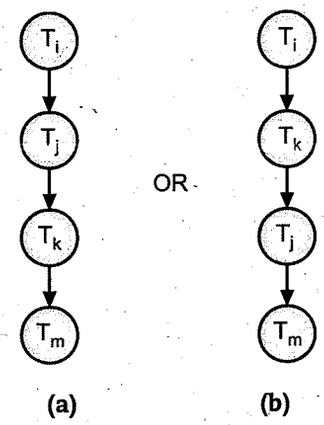


Fig. 3.9

Hence, the schedule corresponding to precedence graph in Fig. 3.9 is conflict serializable.

(vi) Consider the precedence graph in Fig. 3.10.

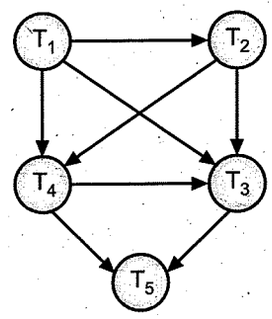


Fig. 3.10

The graph is acyclic. The serial schedules that are conflict equivalent to given schedule are:

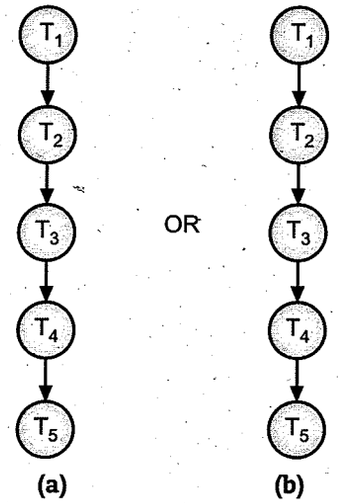


Fig. 3.11

Hence, the schedule corresponding to given precedence graph is conflict serializable.

2. Testing for View serializability:

- Testing for view serializability is complicated. It has been shown that testing for view serializability is itself NP-complete. Thus there exists no algorithm to test for view serializability.
- However, concurrency control schemes can still use sufficient conditions for view serializability. That is if sufficient conditions are satisfied, the schedule is view serializable schedule. But there may be view serializable schedules that do not satisfy the sufficient conditions.

Recoverability of Transaction:

- For recovering a transaction following techniques are used:

1. Recoverable and Non-recoverable Schedules:

- A recoverable schedule is one where for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j. Otherwise the schedule is non-recoverable.
- Consider the schedule given in Table 3.18. Transaction T_9 reads the data written by T_8. Commit of transaction T_8 occurs after commit of transaction T_9. Hence, it is a non-recoverable schedule.

Table 3.18: Schedule 13

T_8	T_9
read(A)	
write(A)	read(A)
read(B)	

2. Cascadeless schedule:

- Even if a schedule is recoverable, to recover correctly from the failure of a transaction T_i, we may have to rollback the transaction. Consider the partial schedule shown below:

Table 3.19: Schedule 14

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	read(A)
	write(A)	

- Transaction T_{10} writes a value of A that is read by transaction T_{11}. Transaction T_{11} writes a value of A that is read by transaction T_{12}. Suppose that at a point transaction T_{10} fails. T_{10} must be rolled back. Since T_{11} is dependent on T_{10}, T_{11} must be rolled back.
- Since, T_{12} is dependent on T_{11}, T_{12} must be rolled back. This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called **cascading rollback.**
- A cascadeless schedule is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before commit operation of T_j.

3.5 LOCK-BASED PROTOCOL AND TIME STAMP-BASED ORDERING PROTOCOLS

Concurrency Control Schemes:

- This section deals with some concurrency control schemes. These schemes ensure that the schedules produced by concurrent transaction are serializable.
- The isolation property of transaction can be preserved by controlling the interaction among the concurrent transactions and this control is done using different mechanisms called as Concurrency Control Mechanisms or Concurrency Control Schemes.
- Following are the concurrency control schemes:
  - Lock-based protocol.
  - Time-stamp based protocol.
  - Validation based protocol.
  - Multiple granularity.
  - Multiversion schemes.

3.5.1 Lock-based Protocol

- Serializability, can easily be ensured if access to database is done in mutually exclusive manner i.e. if one transaction is accessing a data item, no other transaction can modify that data item. The most common method to implement mutual exclusion is to use locks.
- A transaction is allowed to access a data item only if it's currently holding a lock on that item.

1. Lock:

- Consider that database is made up of data-items. A lock is a variable associated with each data item.
- Manipulating the value of lock is called Locking. The value of lock is used in locking schemes to control the concurrent access and to manipulate the associated data items.
- Following are the two types or modes of locks.
  - **(i) Shared:** If a transaction T_i has obtained a shared mode lock (denoted by S) on item A, then T_i can read but it cannot write A.
  - **(ii) Exclusive:** If a transaction T_i, has obtained an exclusive mode lock (denoted by X) on item A, then T_i can both read and write A.
- Depending on the type of the operation, the transaction requests a lock in an appropriate mode on data item.
- The request is made to the concurrency-control manager, and the transaction can proceed with operation only after the concurrency-control manager grants the lock to that transaction.

2. Compatibility Function:

- Given a set of lock modes, the compatibility function can be defined as: 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 currently holds a lock of mode B. If transaction T_i can be granted a lock on Q immediately, in spite of presence of the lock of mode B, then we say that mode A is compatible with B. Such a function can be represented conveniently by a matrix. Compatibility of two modes of lock is given in matrix comp.

Table 3.20: Comp lock-compatibility Matrix

	S	X
S	true	false
X	false	false

- If comp (A, B) is true it to means that A is compatible with B. Notice that shared mode (S) is compatible with shared mode(S). At any time, several shared-mode locks can be held simultaneously on a particular data item.

- A transaction requests a shared lock on data item Q by executing the lock - S(Q) instruction and an exclusive lock is requested through lock-X(Q) instruction. A data item Q can be unlocked via the unlock(Q) instruction.
- Transaction T_i can unlock the data item Q by executing unlock(C) instruction. But transaction must hold a lock on a data item, as long as it accesses the data item.
- Locking protocols indicate when a transaction may lock and unlock each of the data items. Each transaction must follow the set of rules specified by locking protocols. Locking protocols restrict the number of possible serializable schedules. This section deals with only those locking protocols which allow conflict serializable schedules.

3. Conflict Serializability:

- Let $\{T_0, T_1, ..., T_n\}$ be a set of transactions participating in a schedule S. We say that T_i precedes T_j in S, written $T_i \rightarrow T_j$ if there exists a data item Q such that T_i has held lock mode A on Q and T_j has held lock mode B on Q later and comp (A, B) = false. If $T_i \rightarrow T_j$, then that precedence implies that in any equivalent serial schedule, T_i must appear before T_j.
- This graph is similar to the precedence graph. Conflicts between instructions correspond to non-compatibility of lock modes. We say that a schedule S is legal under a given locking protocol if S is possible schedule for a set of transactions following the rules of locking protocol.
- We say that a locking protocol ensures conflict serializability if and only if for all legal schedules the associated $\rightarrow$ relation is acyclic.

4. Starvation of locks:

- Suppose that transaction:

  T_2 - has a shared mode lock on data item, and

  T_1 - requests an exclusive mode lock on same data item.

  Clearly T_1 has to wait for T_2 to release the shared mode lock. Meanwhile, suppose that

  T_3 - request a shared mode lock on same data item.
- This lock request is compatible with the lock granted to T_2, so T_3 may be granted the shared mode lock.
- At this point, T_2 may release the lock but still T_1 has to wait for T_3. But again there may be a new transaction T_4 requesting a shared mode lock on the same data-item. It is possible that there is a sequence of transactions that each requests a shared mode lock on same data item and T_1 never gets the exclusive mode lock on the data item. The transaction T_1 may never make progress and is said to be starved.
- Starvation of transactions can be avoided by granting locks as follows. When a transaction T_i requests a lock on a data item Q in a particular mode M, the lock is granted provided that:
  - (i) There is no other transaction holding a lock on Q in a mode that conflicts with M.
  - (ii) There is no other transaction that is waiting for a lock on Q and that made its lock request before T_i.

5. Locking Protocols:

Now we shall study the locking protocols.

(i) Two-phase locking protocol.

(ii) Graph-based protocol.

(i) Two Phase Locking Protocol:

- This protocol requires that each transaction issue a lock and unlock requests in two phases:
  - **(a) Growing phase:** A transaction may obtain locks, but may not release any lock.
  - **(b) Shrinking phase:** A transaction may release locks, but may not obtain any new locks. Initially the transaction is in growing phase. In this it acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase and it can issue no more 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.
- The transactions can be ordered according to lock points. This ordering gives the serializability ordering for transaction. This serial schedule is conflict equivalent i.e. the two-phase locking protocol ensures conflict serializability.

Example: Following two transactions are two phase transactions.

T_3: lock - X (B)
read (B)
B: = B – 50
write (B)
lock – X(A)
read (A)
A: = A + 50
write (A)
unlock (B)
unlock (A)

T_4: lock-S (A)
read (A)
lock - S(B)
read (B)
display (A + B)
unlock (A)
unlock (B)

- The unlock instructions do not need to appear at the end of the transaction.
- Two phase locking does not ensure freedom from deadlock. Transactions T_3 and T_4 are two phases, but they are deadlocked in the schedule.

Table 3.21: Schedule 15

T_3	T_4
lock-X (B)	
read (B)	
B:= B – 50	
write (B)	
	lock- S (A)
	read (A)
	lock-S (B)
lock-X (A)	
read (A)	
A:= A + 50	
write (A)	
unlock (B)	
unlock (A)	
	read (B)
	display (A + B)
	unlock (A)
	unlock (B)

- Cascading rollback may occur under two phase locking.

Variations on Two Phase Locking:

1. Strict Two Phase Locking Protocol:

Cascading rollbacks can be avoided by a modification of two phase locking called Strict Two Phase Locking Protocol. It requires that in addition to locking being two phases, all exclusive-mode locks taken by a transaction must be held until that transaction commits.

This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

2. Rigorous Two Phase Locking Protocol:

It requires that all locks to be held until the transaction commits. With rigorous two phase locking, transactions can be serialized in the order in which they commit. Most database system implements strict or rigorous two phase locking.

3. Two Phase Locking with Lock Conversion:

Basic two phase locking is modified and lock conversions are allowed. We shall provide a mechanism for upgrading a shared lock to an exclusive lock and downgrading an exclusive lock to shared lock.

Upgrade (Q): Convert shared lock lock-S (Q) to exclusive lock lock-X (Q).

Downgrade (Q): Convert exclusive lock lock-X (Q) to shared lock lock-S (Q).

Lock conversion cannot be allowed to occur arbitrarily. Upgrading can take place in growing phase and downgrading can take place in only the shrinking phase. Two phase locking with lock conversion generates conflict serializable schedule.

Example: Consider the following two transactions with only read and write operations:

T_8: read (a_1)
read (a_2)
read (a_3)
read (a_4)
read (a_5)
write (a_1)

T_9: read (a_1)
read (a_2)
display ($a_1 + a_2$)

- If we employ two phase locking with lock conversion, the schedule can be given as:
  - Automatic generation of appropriate lock and unlock instructions for a transaction is also possible. When a transaction issues a read (a) operation, the system issues lock-S (Q) instruction followed by read (a) operation.
  - When a transaction T_i issues a write (Q) operation, the system checks to see whether T_i already holds a shared lock on Q. If it does then the system issues an upgrade (Q) instruction, followed by the write (Q) instruction. Otherwise, the system issues lock-X (Q) instruction followed by write (Q) operation.

Table 3.22: Schedule 16

T_8	T_9
lock-S (a_1)	
read (a_1)	
	lock-S (a_1)
	read (a_1)

lock-S (a_2)	
read (a_2)	
	lock-S (a_2)
	read (a_2)
lock-S (a_3)	
lock-S (a_4)	
	display ($a_1 + a_2$)
	unlock (a_1)
	unlock (a_2)
lock-S (a_5)	
upgrade (a_1)	
write (a_1)	

(ii) Graph - Based Protocols:

- Graph-based protocol is not a two phase locking protocol and it requires prior knowledge of how each transaction will access the database. To acquire such prior knowledge it uses data graph.
- Let $D = \{d_1, d_2, ..., d_n\}$ is the set of all data items. If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access d_i before accessing d_j. This partial ordering may be a result of either the logical or physical organization of data or it may be imposed solely for the purpose of concurrency control.
- The partial ordering implies that the set D may be viewed as a directed acyclic graph, called database graph. For simplicity we shall consider only those graphs which are rooted trees.
- A simple protocol called tree-protocol, which is restricted to exclusive locks is stated here. Each transaction T_i can lock (lock-X (Q)) a data item at most once and observe the following rules.
  (a) The first lock by T_i may be on any data item.
  (b) Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.
  (c) Data item may be unlocked at any time.
  (d) A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.
- All schedules that are legal under the tree protocol are conflict serializable.

Example: Consider the database graph in Fig. 3.12. The following four transactions follow the tree protocol on this graph.

T_{10}: lock-X (B)
lock-X (E)
lock-X (D)
unlock (B)
unlock (E)
lock-X (G)
unlock (D)
unlock (G)

T_{11}: lock-X (D)
lock-X (H)
unlock (D)
unlock (H)

T_{12}: lock-X (B)
lock-X (E)
unlock (E)
unlock (B)

T_{13}: lock-X (D)
lock-X (H)
unlock (D)
unlock (H)

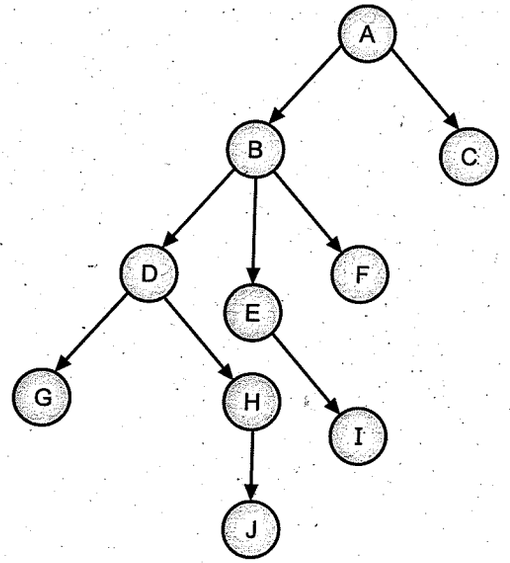


Fig. 3.12: Tree-structured Database graph

- The schedule following tree protocol is given below:

Table 3.23: Schedule for database graph in Fig 3.12

T_{10}	T_{11}	T_{12}	T_{13}
lock-X (B)			
	lock-X (D)		
	lock-X (H)		
	unlock (D)		
lock-X (E)			
lock-X (D)			
unlock (B)			
unlock (E)			
		lock-X (B)	
		lock-X (E)	
	unlock (H)		
lock-X (G)			
unlock (D)			
			lock-X (D)
			lock-X (H)
			unlock (D)
			unlock (H)
		unlock (E)	
		unlock (B)	
unlock (G)			

- This schedule is conflict serializable. Tree protocol ensures conflict serializability and it is free from deadlocks.

Advantages:

- Tree locking protocol has an advantage over two phase locking protocol:

1. Unlocking may occur earlier. Earlier unlocking may lead to shorter waiting times and to an increase in concurrency.
2. It is deadlock free and hence no rollbacks are required.

Disadvantage:

1. In some cases, a transaction may have to lock data items that it does not access.

- **For Example,** A transaction that needs to access data items A and J in the database graph given in Fig. 3.12 must lock not only A and J but also data items B, D and H.

- These additional locking results in increased locking overhead. The possibility of additional waiting time and potential decrease in concurrency. Further, without prior knowledge of what data item will need to be locked, transactions will have to lock the root of tree and that can reduce the concurrency greatly.

3.5.2 Timestamp Based Protocol

Timestamp:

- A **timestamp** is a unique identifier created by database management system to identify a transaction.
- With each transaction T_i in the system, a fixed value called timestamp is associated, denoted by
  $T_s(T_i)$. This timestamp is assigned by database system before T_i starts execution.
- If transaction T_i is assigned a timestamp $T_s(T_i)$ and a new transaction T_j enters the system, then $T_s(T_i) < T_s(T_j)$.
- There are two simple methods for implementing timestamp scheme.
  1. Use the value of system clock as the timestamp i.e. a transaction's timestamp is equal to the value of system clock, when the transaction enters the system.
  2. Use a logical counter that is incremented after a new timestamp has been assigned i.e. a transaction's timestamp is equal to the value of the counter when the transaction enters the system.
- Timestamp of transaction determines the serializability order. If $T_s(T_i) < T_s(T_j)$ then the system must ensure that the produced schedule is equivalent to serial schedule in which T_i appears before T_j.
- To implement this scheme, two timestamp values are associated with each data item Q.
  1. **W-Timestamp (Q):** It denotes the largest timestamp of any transaction that executed write (Q) successfully.
  2. **R-Timestamp (Q):** It 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.

Timestamp Ordering Protocol:

- This protocol decides the ordering of transactions in advance to determine the serializability order. For this it uses timestamp ordering scheme.
- This protocol operates as follows:
  1. **Suppose that transaction T_i issues read (Q).**
     (i) If $T_s(T_i) <$ 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.

(ii) If $T_s(T_i) \geq$ W-Timestamp (Q), then the read operation is executed, and R-Timestamp (Q) is set to maximum of R-Timestamp (Q) and $T_s(T_i)$.

2. **Suppose that T_i issues write (Q).**
   (i) If $T_s(T_i) <$ R-Timestamp (Q) then the value of Q that T_i is producing was needed previously and the system assumed that value would never be produced. Hence, the write operation is rejected, and T_i is rolled back.
   (ii) If $T_s(T_i) <$ W-timestamp (Q), then T_i is attempting to write an obsolete value of Q. Hence, this write operation is rejected, T_i is rolled back.
   (iii) Otherwise, the write operation is executed and W-timestamp is set to $T_s(T_i)$.

- A transaction T_i, that is rolled back by the concurrency control scheme as a result of either a read or write operation being issued, is assigned a new timestamp and is restarted.

Example: Consider the given transactions T_{14} and T_{15}.

T_{14} : read (B)
read (A)
display (A + B)

T_{15} : read (B)
B:= B − 50
write (B)
read (A)
A:= A + 50
write (A)
display (A + B)

- Concurrent schedule for the two transactions is given below. Here, we consider that $T_s(T_{14}) < T_s(T_{15})$. The schedule given in figure is possible under the timestamp protocol.

Table 3.24: Concurrent schedule

T_{14}	T_{15}
read (B)	
	read (B)
	B: = B − 50
	write (B)
read (A)	
	read (A)
display (A + B)	
	A: = A + 50
	write (A)
	display (A + B)

(i) read (B) of T_{14} is executed because T_s (T_{14}) ≥ W-timestamp (B) and it sets R-Timestamp (B) = T_s (T_{14}).

(ii) read (B) of T_{15} is executed because T_s (T_{15}) ≥ W-timestamp (B) and it sets R-Timestamp (B) = T_s (T_{15}).

(iii) write (B) of T_{15} is executed because T_s (T_{15}) = R-Timestamp (B).

Similarly, read (A) of T_{14} and write (A) of T_{15} are executed.

- **Thomas Write Rule:** Consider the schedule given in Table 3.25.

Table 3.25: Schedule 17

T_{16}	T_{17}
read (Q)	
	write (Q)
write (Q)	

Apply Timestamp-ordering protocol to given schedule.

- Since, T_{16} starts before T_{17}, T_j (T_{16}) < T_i (T_{17}), read (Q) of T_{16} and write (Q) operations succeed. When T_{16} attempts its write (Q) operation, T_s (T_{16}) < W-Timestamp (Q) since W-Timestamp = T_{17}. According to Timestamp protocol, write (Q) must be rejected, T_{16} will be rolled back.
- Timestamp ordering protocol rolls back the transaction T_{16}, but the value of write (Q) Operation of T_{16} is already written by write (Q) of T_{17}, and the value that write (Q) of T_{16} is attempting to write will never be read i.e. we can ignore the write (Q) of T_{16}.
- This leads to a modification of Timestamp-ordering protocol. This modified protocol operates as follows:

1. **Suppose that transaction T_i issues read (Q):**
   (i) If T_s (T_i) < W-Timestamp (Q), then T_i needs to read a value of Q that was already over written. Hence, the read operation is rejected, and T_i is rolled back.
   (ii) If T_s (T_i) ≥ W-timestamp (Q), then read operation is executed, and R-timestamp (Q) is set to the maximum of R-timestamp (Q) and T_s (T_i).
2. **Suppose that transaction T_i issues write (Q):**
   (i) If T_s (T_i) < R-Timestamp (Q) then the value of Q that T_i is producing was previously needed, and it was assumed that the value would never be produced. Hence, the write operation is rejected, and T_i is rolled back.
   (ii) If T_s (T_i) < W-timestamp (Q), then T_i is attempting write an obsolete value of Q. Hence, the write operation can be ignored.
   (iii) Otherwise, the write operation is executed, and W-timestamp (Q) is set to T_s (T_i). i.e. here if T_s (T_i) < W-Timestamp (Q), then we ignore the absolute write operation. This modification to timestamp ordering protocol is called Thomas Write Rule.

- Thomas Write Rule for schedule given in table 3.25 results in a serial schedule < T_{16}, T_{17} > which is view equivalent to given schedule.

3.6 DEADLOCK HANDLING

- Let us see what deadlock state is and how to prevent deadlocks.
- A system is said in a **deadlock state** if every transaction is waiting for other transaction to unlock the database.
- A system is in deadlock state if there exist a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- If {T_0, T_1, ..., T_n} is the set of transactions such that T_0 is waiting for a data item that is held by T_1 and T_1 is waiting for a data item that is held by T_2 ... and T_{n-1} is waiting for a data item that is held by T_n. T_n is waiting for a data item that is held by T_0. Hence, none of the transactions can make progress in such situation. That is the system is in deadlock state.

3.6.1 Methods for Dealing with Deadlock Problems

- There are three methods for dealing with deadlock problems:
  1. Deadlock prevention.
  2. Time-out based schemes.
  3. Deadlock detection and deadlock recovery.

3.6.1.1 Deadlock Prevention

- There are two approaches to deadlock prevention:
  1. It ensures that no cyclic waits can occur by ordering the requests for locks or requiring all locks to be acquired together.
  2. It performs transaction rollbacks instead of waiting for a lock, whenever the wait could potentially result in a deadlock.

Following are the schemes under first approach:

- Lock all the data items before a transaction begins its execution. But there are two main disadvantages of using this protocol:
  1. It is often hard to predict, before the transaction, 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.
- Another scheme is to impose a partial ordering of all at data items, and transaction can lock the data items only in that order. Using tree protocol, this scheme can be implemented.

Following are the schemes for second approach:

- The second approach uses pre-emption and transaction rollbacks. When a transaction T_2 requests a lock that is held by a transaction T_1, the lock granted to T_1, may be pre-empted by rolling back of T_1 and granting of lock to T_2.
- To control the pre-emption, we assign a unique timestamp to each transaction. These timestamps will be used to decide whether the transaction should wait or rollback.

Locking is used for concurrency control. If a transaction is rolled back, it retains the old timestamp when restarted.

- **Two different deadlock prevention schemes using timestamps under the second approach are:**

1. **Wait-Die:** Scheme is based on non-preemptive technique. When a transaction T_i requests a lock on 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. Otherwise T_i is rolled back (die).

   Example: $T_s(T_i) = 5$

   $T_s(T_j) = 10$

   $T_s(T_k) = 15$

   If T_i requests a lock on a data item held by T_j then T_i will wait since $T_s(T_i) < T_s(T_j)$. If T_k requests a lock on a data item held by T_j, then T_k will be rolled back since $T_s(T_k) > T_s(T_j)$.

2. **Wound-wait:** This scheme is based on preemptive technique.

   When a transaction T_i requests a lock on 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. Otherwise, T_j is rolled back (T_j is wounded by T_i).

   Example: Consider the timestamps given in previous example for T_i, T_j and T_k transactions. If T_i requests a lock on data item held by T_j, then T_j will be rolled back.

   $$T_s(T_i) < T_s(T_j)$$

   If T_k requests a lock on a data item held by T_j, then T_k will wait since

   $$T_s(T_k) > T_s(T_j)$$

Comparison of wait-die and wait-wound schemes:

1. Both the wait-die and wait-wound schemes avoid starvation.
2. In wait-die scheme, an older transaction must wait for younger one to release the data item. Whereas in wait-wound scheme, the older transaction never waits for a younger transaction.
3. Number of rollbacks in wait-wound scheme is fewer as compared to wait-die scheme.
4. Major problem in both the schemes is unnecessary rollbacks.

3.6.1.2 Timeout Based Schemes

- In this approach deadlock handling is based on lock time-outs.
- A transmission that has requested a lock waits for at most a specified amount of time.
- If the lock has not been granted within that time, the transaction is said to time out and it rolls back itself back and restarts.
- If there was a deadlock, one or more transactions involved in deadlock will time-out and rollback allowing others to proceed.

Advantages and Disadvantages:

1. It is easy to implement.
2. It works well if transactions are short and if long waits are likely to be due to deadlocks.
3. It is difficult to decide how long a transaction must wait before time-out. Too long wait results in unnecessary delays once a deadlock has occurred. Too short wait results in transaction rollbacks even when there is no deadlock, leading to wasted resources.
4. Starvation is the possibility with this scheme.

3.6.1.3 Deadlock Detection and Recovery

- This is one method for dealing with deadlock. It allows the system to enter a deadlock state, and then try to recover using deadlock detection and deadlock recovery scheme. If the probability that system enters deadlock state is relatively low, this method is efficient.
- An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred. If it has occurred then the system must attempt to recover from the deadlock.
- Following are the requirements for deadlock detection and recovery:
  a. Maintain information about the current allocation of data items to transactions as well as any outstanding data item requests.
  b. Provide an algorithm that uses this information to determine whether the system has entered a deadlock state.
  c. Recover from the deadlock when detection algorithm determines that a deadlock exists.

1. **Deadlock Detection:**

- Deadlocks can be detected using a directed graph called **Wait for Graph**.
- The wait for graph consists of a pair G = (V, E) where V is a set of *vertices* and E is a set of *edges*. The set of vertices consists of all transactions in the system.
- Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$.
- A directed edge $T \rightarrow T_j$ in graph implies that T_i is waiting for transaction T_j to release the data item. The edge $T_i \rightarrow T_j$ is removed when T_j is no longer holding a data item needed by transaction T_i.
- A deadlock exists in the system if the wait for graph for that system contains a cycle.
- Consider the Wait Graph given in Fig. 3.13.

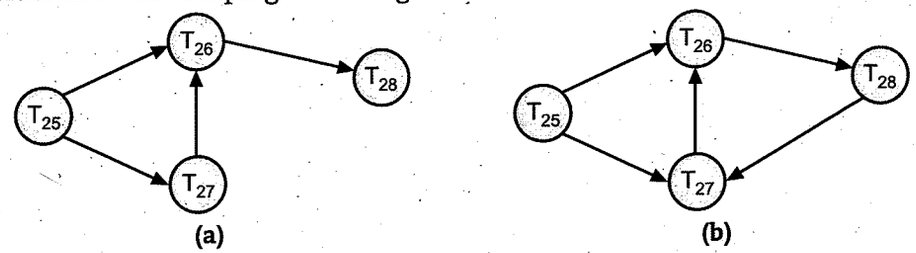


Fig. 3.13: Wait Graph

Transaction T_{25} is waiting for T_{26} and T_{27}.

Transaction T_{26} is waiting for T_{28}.

Transaction T_{27} is waiting for T_{26}.

The graph contains no cycle. Hence, the system is not in deadlock state.

But now consider the Fig. 3.13 (b). It contains a cycle $T_{26} \rightarrow T_{28} \rightarrow T_{27}$

- Hence, the system is in deadlock state. In this way using this algorithm deadlocks can be detected. Depending on the frequency of deadlock occurrence, the deadlock detection algorithm should be invoked.

2. Recovery from Deadlock:

- The most common solution to recover from deadlock is to rollback one or more transactions to break the deadlock.
- A transaction can be recovered using following three actions.

(i) Selection of victim:

- Determine which transaction to rollback. Those transactions that will incur the minimum cost will be rolled back.
- The cost of rollback can be decided by following factors.
  (a) How long the transaction has computed and how much longer the transaction will compute before it completes the designated task?
  (b) How many data items it has used?
  (c) How many more data items it needs to complete?
  (d) How many transactions will be involved in the rollback?

(ii) Rollback:

- One method to rollback a transaction is to abort transaction and restart it.
- The other more effective method is to rollback the transaction only as far as necessary to break the deadlock. But this requires additional information about all the running transactions.

(iii) Starvation:

- It may happen that the same transaction is always selected as victim. This results in starvation.
- The most common solution is to include the number of rollbacks in the cost factor.

Summary

- A transaction is a logical unit of processing in a DBMS which entails one or more database access operation.
- Not managing concurrent access may create issues like hardware failure and system crashes.
- Active, Partially Committed, Committed, Failed and Terminate are important transaction states.

- ACID stands for Atomicity, Consistency, Isolation, and Durability.
- A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one.
- Serializability is the process of search for a concurrent schedule which output is equal to a serial schedule where transaction execute one after the other.
- Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another.
- Lost Updates, dirty read, Non-Repeatable Read, phantom read and Incorrect Summary Issue are problems faced due to lack of concurrency control.
- Two-Phase locking protocol which is also known as a 2PL protocol needs transaction should acquire a lock after it releases one of its locks. It has 2 phases growing and shrinking.
- The timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. The protocol uses the System Time or Logical Count as a Timestamp.
- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks.
- There are three methods for dealing with deadlock problems: Deadlock prevention, Time-out based schemes, Deadlock detection and deadlock recovery.

Check Your Understanding

1. Which of the following locks the item from access of any type?
   (a) Implicit lock (b) Explicit lock
   (c) Exclusive locks (d) Shared lock
2. What are ACID properties of Transactions?
   (a) Atomicity, Consistency, Isolation, Database
   (b) Atomicity, Consistency, Isolation, Durability
   (c) Atomicity, Consistency, Inconsistent, Durability
   (d) Automatically, Concurrency, Isolation, Durability
3. Database locking concept is used to solve the problem of _____.
   (a) Lost Update (b) Uncommitted Dependency
   (c) Inconsistent Data (d) All of the above
4. Term stating either all operations of transaction to be displayed at database or none at all is known to be _____.
   (a) Atomicity (b) Inconsistency
   (c) Isolation (d) Durability

5. A protocol that ensures system will never enter a deadlock state is called _____.
   (a) Deadlock detection (b) Deadlock elimination
   (c) Deadlock prevention (d) Deadlock recovery
6. The protocol that indicates when a transaction may lock and unlock each of the data items is called as _____
   (a) Locking protocol (b) Unlocking protocol
   (c) Granting protocol (d) Conflict protocol
7. Which of the following occurs when one transaction reads a changed record that has not been committed to the database?
   (a) Nonrepeatable read (b) Phantom read
   (c) Dirty read (d) Consistent read
8. Which of the following is not a deadlock handling strategy?
   (a) Deadlock prevention (b) Timeout
   (c) Deadlock detection and recovery (d) Deadlock filtration
9. Consider the following two statements about database transaction schedules:
   I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.
   II. Timestamp-ordering concurrency control protocol with Thomas' Write Rule can generate view serializable schedules that are not conflict serializable.
   Which of the above statements is/are TRUE?
   (a) I only (b) II only
   (c) Both I and II (d) Neither I nor II
10. Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock?
    I. 2-phase locking
    II. Time-stamp ordering
    (a) I only (b) II only
    (c) Both I and II (d) Neither I nor II

ANSWER KEY

1. (c)	2. (b)	3. (d)	4. (a)	5. (c)
6. (a)	7. (c)	8. (d)	9. (c)	10. (b)

Practice Questions

Q.I: Answers the following questions in short.

1. What is meant by concurrency control?

2. List the ACID properties. Explain usefulness of each.
3. What is serializability? Explain its types.
4. Define the following terms:
   a. Timestamp
   b. Lock
5. What is deadlock state?
6. Which methods are used to handle deadlocks?

Q.II: Answers the following questions.

1. What is transaction? Explain the desirable properties of a transaction.
2. Differentiate between serial schedule and serializable schedule.
3. With the help of diagram describe states of transaction.
4. What is deadlock? How to detect and recover it? Explain with example.
5. What is lock in DBMS? Explain two phase locking protocol for concurrency control.
6. Explain state of transaction with suitable diagram. Explain conflict serializability and view serializability with suitable example.
7. Consider the following two transactions:

   T1: read(A)
   read(B)
   if A = 0 then B:= B + 1
   write(B)
   T2: read(B)
   read(A)
   if B = 0 then A:= A + 1
   write(A)
   Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$

   - Show that every serial execution involving these two transactions preserves the consistency of the database.
   - Show a concurrent execution of T_1 and T_2 that produces a non-serializable schedule.

Q.III: Write a short note on:

1. Two-phase commit protocol
2. Timestamp based protocol
3. Deadlock handling
4. Serializability
5. Concurrency control

❖❖❖

4... Parallel Databases

Objectives...

- To introduce the concept of Parallel Databases.
- To learn about Parallel Database Architectures.
- To study various Parallelisms.
- To know about key elements of Parallel database processing.

4.1 INTRODUCTION TO PARALLEL DATABASES

- Parallel database system improves performance of data processing using multiple resources in parallel, like multiple CPU and disks are used in parallel manner. It also performs many parallelization operations like, data loading and query processing.
- Parallel databases are especially useful for applications that have to query large databases and process large number of transactions per second.

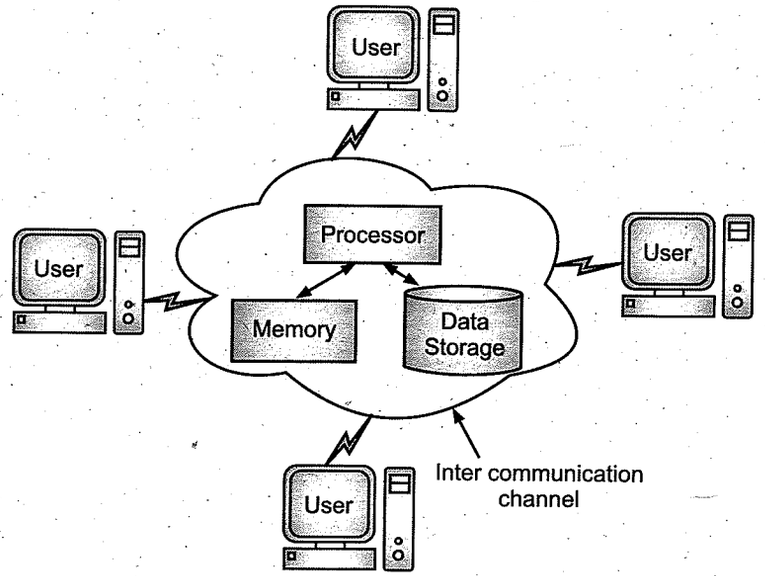


Fig. 4.1: Parallel Database System

- Parallel database systems are usually designed from the ground up to provide best cost-performance and they are quite uniform in site machine (computer) architecture. The co-operation between site machines is usually achieved at the level of the transaction manager module of a database system.
- Parallel database systems represent an attempt to construct a faster centralised computer using several small CPUs. It is more economical to have several smaller CPUs that together have the power of one large CPU.

Advantages of Parallel Databases:

- There are many advantages of parallel databases. Some of these are as follows:
  1. **Improve performance:** The performance of the system can be improved by connecting multiple CPU and disks in parallel. Many small processors can also be connected in parallel.
  2. **Improve availability of data:** Data can be copied to multiple locations to improve the availability of data. For example, if a module contains a relation (table in database) which is unavailable then it is important to make it available from another module.
  3. **Improve reliability:** Reliability of system is improved with completeness, accuracy and availability of data.
  4. **Provide distributed access of data:** Companies having many branches in multiple cities can access data with the help of parallel database system.

Disadvantages of Parallel Databases:

- There are many disadvantages of parallel databases. Some of them are as follows:
  1. **High cost:** The start-up cost is comparatively high.
  2. **Data skew:** Skew causes processor's dealing with large partitions of multiple disks to become performance bottleneck.
  3. **Inference problem:** As more CPU's are added, existing CPU's get slow down due to increased contention of memory and network bandwidth.

4.2 PARALLEL DATABASE ARCHITECTURES

- Parallel database architectures can be broadly classified into three categories: Shared Memory, Shared Disk, and Shared Nothing.
- Following figures (Fig. 4.2, 4.3 and 4.4) shows the different architecture proposed and successfully implemented in the area of Parallel Database systems. In the following figures, P represents Processors, M represents Memory, and D represents Disks/Disk setups.

1. Shared Memory Architecture:

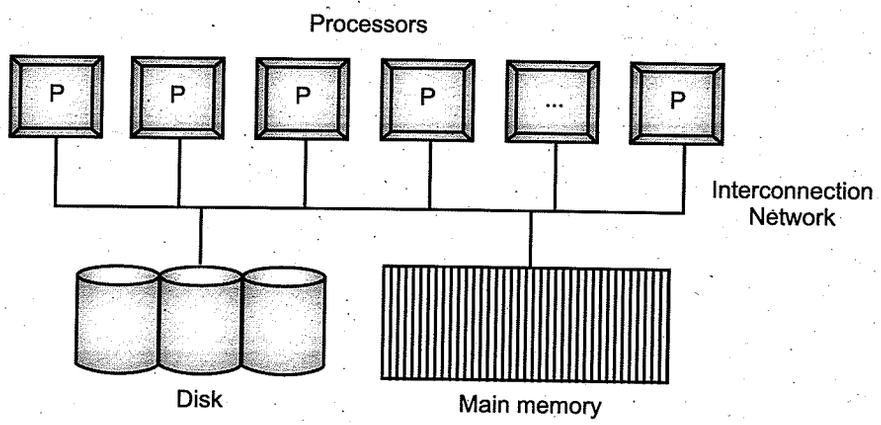


Fig. 4.2: Shared Memory Architecture

- In Shared Memory architecture, single memory is shared among many processors as shown in Fig. 4.2. Several processors are connected through an interconnection network with Main memory and disk setup. Here interconnection network is usually a high speed network (may be Bus, Mesh, or Hypercube) which makes data sharing (transporting) easy among the various components (Processor, Memory, and Disk).
- Note that, shared memory architecture is a tightly coupled architecture in which multiple processors share memory. It is also known as Symmetric Multiprocessing (SMP).

Advantages:

1. Simple implementation.
2. Establishes effective communication between processors through single memory addresses space.

Disadvantages:

1. Higher degree of parallelism (more number of concurrent operations in different processors) cannot be achieved due to the reason that all the processors share the same interconnection network to connect with memory. This causes Bottleneck in interconnection network (Interference), especially in the case of Bus interconnection network.
2. Addition of processor would slow down the existing processors.
3. Cache-coherency should be maintained. That is, if any processor tries to read the data used or modified by other processors, then we need to ensure that the data is of latest version.
4. Degree of Parallelism is limited. More number of parallel processes might degrade the performance.

2. Shared Disk Architecture:

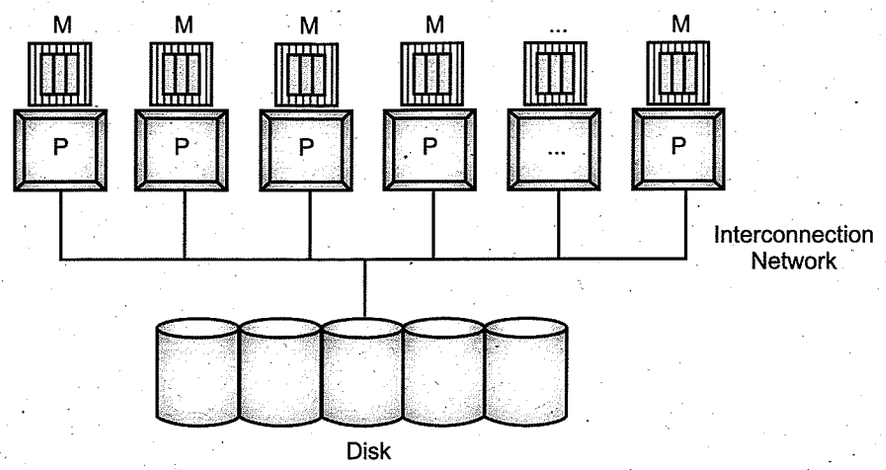


Fig. 4.3: Shared Disk Architecture

- In Shared Disk architecture, single disk or single disk setup is shared among all the available processors and also all the processors have their own private memories as shown in Figure 4.3.
- It is loosely coupled architecture optimized for application that is inherently centralized.
- Each processor can access all disks directly but each has its own private memory. So, Shared disk systems are sometimes referred to as clusters.

Advantages:

1. Failure of any processors would not stop the entire system (Fault tolerance).
2. Interconnection network is no longer a bottleneck as each CPU has its own memory.
3. Support larger number of processors (when compared to Shared Memory architecture).

Disadvantages:

1. When the number of CPUs increases, problems of interference and memory contentions also increases.
2. Inter-processor communication is slow. The reason is all the processors have their own memory. Hence, the communication between processors need reading of data from other processors' memory which needs additional software support.

Example of Real Time Shared Disk Implementation:

- DEC clusters (VMS cluster) running Rdb.

3. Shared Nothing Architecture:

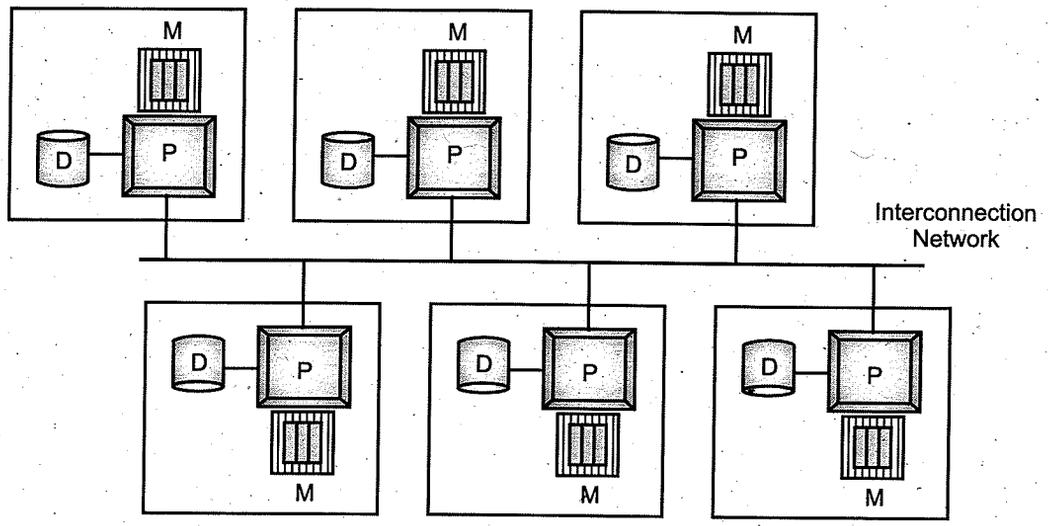


Fig. 4.4: Shared Nothing Architecture

- In Shared Nothing architecture, every processor has its own memory and disk setup. This setup may be considered as set of individual computers connected through high speed interconnection network using regular network protocols and switches, for example to share data between computers.
- Two Processors can not access the same disk area. There is no sharing of memory or disk resources.
- Each processor has its own copy of OS and DBMS. This architecture is also known as Massively Parallel Processing (MPP).

Advantages:

1. Number of processors used here is scalable. That is, the design is flexible to add more number of computers.
2. Unlike in other two architectures, only the data request which cannot be answered by local processors need to be forwarded through interconnection network.

Disadvantages:

1. Non-local disk accesses are costly. That is, if one server receives the request. If the required data not available, it must be routed to the server where the data is available. It is slightly complex.
2. Communication cost involved in transporting data among computers.

Examples of Real Time Shared Nothing Implementation:

- Teradata
- Tandem
- Oracle nCUBE

4.3 DATABASE PARALLELISM

- Database Parallelism is a method of implementing parallel processing in a database. Parallelism in the database covers all the operations that are usually going through in a database, like loading the data, transforming the data, executing the queries, etc. concurrently. This helps to improve data consistency, system performance, parallel access of data, by making use of multiple data sources, multiple system's memory usages, multiple disk space occupation, Hierarchical flow with lesser time spent compared to individual process execution.

Methods of Parallelism:

- Parallelism has been achieved by the following methods:
  1. I/O Parallelism
  2. Intra-query Parallelism
  3. Inter-query Parallelism
  4. Intra-operational Parallelism
  5. Inter-operational Parallelism
- Let us see details of these methods:

4.3.1 I/O Parallelism

- Input/Output (I/O) parallelism is the simplest form of parallelism in which the relations (tables) are partitioned on multiple disks to reduce the retrieval time of relations from disk. In I/O parallelism, the input data is partitioned and then each partition is processed in parallel. These results are combined after the processing of all partitioned data. I/O parallelism is also called data partitioning.
- The following types of partitioning techniques can be used:
  1. Hash partitioning.
  2. Range partitioning.
  3. Round-robin partitioning.
  4. Schema partitioning.

4.3.1.1 Hash Partitioning

- In the technique of hash partitioning a hash function is applied to the attribute value whose range is {0, 1, 2, ..., n -1}. Each tuple (row) of the original relation is hashed on the partitioning attributes. The output of this function causes the data for that tuple to be targeted for placement on a particular disk.
- For example, let us assume that there are n disks d1, d2 , d3 ,.... dn , across which the data are to be partitioned. Now, if the hash function returns 2, then the tuple is placed on disk d2.

Advantages:

1. Hash partitioning has the advantage of providing for even distribution of data across the available disk, helping to prevent skewing. Skew can slow the performance caused by one or more CPUs and disks getting more work than others.
2. Hash partitioning is best suited for point queries (involving exact matches) based on the partitioning attribute. For example, if a relation is partitioned on the employee identification numbers (EMP_ID), then we can answer the query "Find the record of the employee with employee identification number = 100019" using SQL statement as follows:

```
SELECT * FROM EMPLOYEE WHERE EMP_ID = 100019;
```

3. Hash partitioning is also useful for sequential scans of the entire relation (table) placed on n number of disks. The time taken to scan the relation is approximately 1/n of the time required to scan the relation in a single disk system.

Disadvantages:

1. Hash partitioning technique is not well suited for *point queries* that are based on non-partitioning attributes.
2. It is also not well suited for *range queries*, since typically hash functions do not preserve proximity within a range. For example, hash partitioning will not perform for queries involving range searches such as:

```
SELECT * FROM EMPLOYEE WHERE EMP_ID > 105050 and EMP_ID < 150050;
```

In such case, the search would have to involve most of the disks over which the relation has been partitioned.

4.3.1.2 Range Partitioning

- In the technique of range partitioning an administrator specifies that attribute-values within a certain range are to be placed on a certain disk. In other words, range partitioning distributes contiguous attribute-value ranges to each disk.
- For example, range partitioning with three disks numbered as 0, 1, 2, ..., *n* might place tuples for employee numbers with up to 100000 on disk 0, tuples for employees identification numbers 100001 – 150000 on disk 1, tuples for employee 150001–200000 on disk 2 and so forth.

Advantages:

1. Range partitioning involves placing tuples containing attribute values that fall within a certain range on a disk. This offers good performance for range-based queries and also provides reasonable performance for exact-match (point) queries involving the partitioning attribute.
2. For point queries, the partitioning vector can be used to locate the disk where the tuples reside.

3. For range queries, the partitioning vector is used to find the range of disks on which the tuples may reside.
4. In both cases, the search narrows to exactly those disks that might have any tuples of interest.

Disadvantages:

1. Range partitioning can cause skewing in some cases. For example, consider an EMPLOYEE relation that is partitioned across disk according to employee identification numbers. If tuples containing numbers 100000–150000 are placed on disk 0 (d0) and tuples containing numbers 150001–200000 are placed on disk 1 (d1), then data will be evenly distributed. If the company employs 200000 employees. However, if the company employs only 160000 employees currently and most are assigned numbers 100000–150000, the bulk of the tuples for this relation will be skewed towards disk 0 (d0).

4.3.1.3 Round Robin Partitioning

- In this technique, the relations are scanned in any order and the i^{th} row of the relation is sent to the disk number di Mod n. This technique partitions the relation tuples equally in all the disks i.e., the tuples are distributed evenly among the various disks.
- For example, a system with n disks would place tuple A on disk 0 (d0), tuple B on disk 1 (d1), tuple C on disk 2 (d2) and so forth.

Advantages:

1. It is well suited for the applications that read the entire relation sequentially for each query.
2. The distribution of tuples among the disks are even.

Disadvantage:

1. Both point queries (where specified value required) and range queries (value lies within range) are very difficult to process, since all the n disks are required to process the query.

4.4 INTER-QUERY AND INTRA-QUERY PARALLELISM

- There are two types of query parallelism: Inter-query parallelism and Intra-query parallelism.

4.4.1 Inter-query Parallelism

- In inter-query parallelism, multiple queries (Transactions) execute in parallel at different CPU's. This type of parallelism increases the transaction throughput. The response time of individual transactions is almost same as if they were run in isolation. The basic reason of using the inter-query parallelism is to scale up a transaction processing system.

Advantages:

1. It the easiest form of parallelism to support in a database system, particularly in shared memory parallel system.
2. It increases the transaction throughput.
3. It scales-up a transaction processing system to support a larger number of transactions per second.

Disadvantages:

1. The response time of individual transaction remains almost same as if the transactions were run in isolation.
2. It is more complicated to support in a shared disk or share nothing architecture.
3. It does not help in speeding up long running queries, since each query runs sequentially.

4.4.2 Intra-query Parallelism

- In Intra-query parallelism, a single-query is executed in parallel on multiple CPUs and disks. It speeds-up the long running queries. The individual query can be parallized by parallelizing the individual operations involved in the query.
- There are two main ways by which a single query can be parallelized. These are Intra-operational parallelism and Inter-operational parallelism.

Advantages:

1. It speeds up long running queries.
2. It is useful in decision support application.

- Inter-query is sometimes also called *parallel transaction processing* while Intra-query parallelism is sometimes called *parallel query processing.*

4.5 INTER-OPERATIONAL AND INTRA-OPERATIONAL PARALLELISM

4.5.1 Intra-operational Parallelism

- In intra-operational parallelism, we parallelise the execution of each individual operation of a task, such as sorting, projection, join and so on.
- Since the number of operations in a typical query is small, compared to the number of tuples processed by each operation, intra-operational parallelism scales better with increasing parallelism.

Advantages:

1. Intra-operational parallelism is natural in a database.
2. Degree of parallelism is quite high.

4.5.2 Inter-operational Parallelism

- In inter-operational parallelism, the different operations in a query expression are executed in parallel. The following two types of inter-operational parallelism are used:
  - Pipelined Parallelism
  - Independent Parallelism

4.5.2.1 Pipelined Parallelism

- In pipelined parallelism, the output tuples of one operation *A* are consumed by a second operation *B*, even before the first operation has produced the entire set of tuples in its output. Thus, it is possible to run operations *A* and *B* simultaneously on different processors (CPUs), so that operation *B* consumes tuples in parallel with operation *A* producing them.
- The major advantage of pipelined parallelism in a sequential evaluation is that we can carry out a sequence of such operations without writing any of the intermediate results to disk.

Advantages:

1. Pipelined parallelism is useful with a small number of CPUs. Also, pipelined executions avoid writing intermediate results to disk.

Disadvantages:

1. Pipelined parallelism does not scale up well. First, pipeline chains generally do not attain sufficient length to provide a high degree of parallelism.
2. Second, it is not possible to pipeline relational operators that do not produce output until all inputs have been accessed.
3. Third, only marginal speed-up is obtained for the frequent cases in which one operator's execution cost is much higher than are those of the others.

4.5.2.2 Independent Parallelism

- In an independent parallelism, the operations in a query expression that do not depend on one another can be executed in parallel.

Advantage:

1. Independent parallelism is useful with a lower degree of parallelism.

Disadvantages:

1. Like pipelined parallelism, independent parallelism does not provide a high degree of parallelism.
2. It is less useful in a highly parallel system.

4.6 KEY ELEMENTS OF PARALLEL DATABASE PROCESSING

- There are four key elements of any parallel database processing. They are as follows:
  1. Speed-up
  2. Scale-up
  3. Synchronization
  4. Locking

- Let us define them one by one now.

1. Speed-up:

- Speed-up is a property in which the time taken for performing a task decreases in proportion to the increase in the number of CPUs and disks in parallel. In other words, speed-up is the property of running a given task in less time by increasing the degree of parallelism (more number of hardware). With additional hardware, speed-up holds the task constant and measures the time saved.

  Mathematically,

$$S_k = \frac{T_s}{T_p}$$

  Where,

  S_k - Speed-up

  T_s - Time serial execution

  T_p - Time parallel execution

- Fig. 4.5 illustrates linear and sub-linear speed-up curve of the parallelism. The speed-up curve shows how, for a fixed database size, more transactions can be executed per second by adding more number of resources such as CPUs and disks.

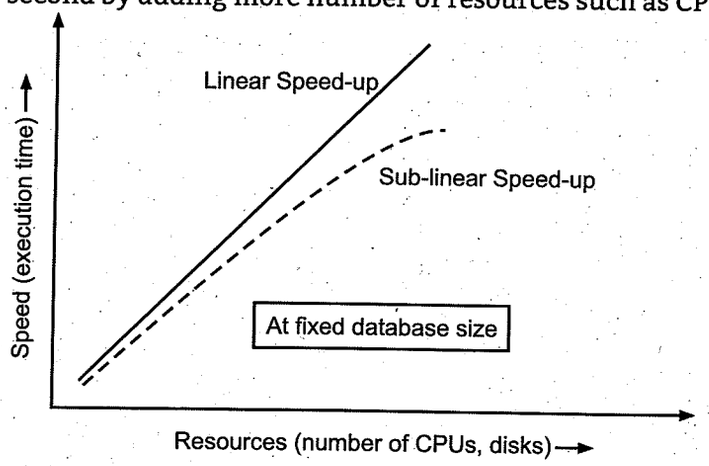


Fig. 4.5: Speed-up with increasing resources

2. Scale-up:

- Scale-up is the property in which the performance of the parallel database is sustained if the number of CPU and disks are increased in proportion to the amount of data. In other words, scale-up is the ability of handling larger tasks by increasing the degree of parallelism (providing more resources) in the same time period as the original system. With added hardware (CPUs and disks), a formula for scale-up holds the time constant and measures the increased size of the task, which can be performed. Thus, scale-up enables users to increase the sizes of their databases while maintaining roughly the same response time.

- Mathematically,

$$\text{Scale-up} = \frac{V_p}{V_o}$$

  Where,

  V_p - Parallel or large possessing volume

  V_o - Original or small possessing volume

- Fig. 4.6 illustrates Linear and Sub-linear scale-up curve of the parallelism.
- The scale-up curve shows how, adding more resources (CPUs) enable the user to process larger tasks. The first scale-up curve measures the number of transactions executed per second as the database size is increased and the number of CPUs is correspondingly increased. An alternative way to measure scale-up is to consider the time taken per transaction (execution time) as more CPUs are added to process an increasing number of transactions per second.

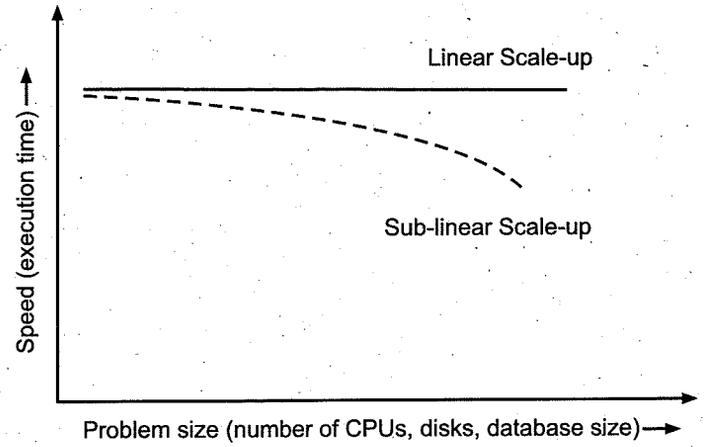


Fig. 4.6: Scale-up with increasing resources

3. Synchronization:

- Synchronisation is the coordination of concurrent tasks. For a successful operation of the parallel database systems, the tasks should be divided such that the synchronisation requirement is less. It is necessary for correctness. With less synchronisation requirement, better speed-up and scale-up can be achieved.
- The amount of synchronisation depends on the amount of resources (CPUs, disks, memory, databases, communication network and so on) and the number of users and tasks working on the resources.
- More synchronisation is required to coordinate large number of concurrent tasks and less synchronisation is necessary to coordinate small number of concurrent tasks.

4. Locking:

- Locking is a method of synchronising concurrent tasks. Both internal as well as external locking mechanisms are used for synchronisation of tasks that are required by the parallel database systems.

- For external locking, a Distributed Lock Manager(DLM) is used, which is a part of the operating system software. DLM coordinates resource sharing between communication nodes running a parallel server.
- The instances of a parallel server use the DLM to communicate with each other and coordinate modification of database resources.
- The DLM allows applications to synchronise access to resources such as data, software and peripheral devices, so that concurrent requests for the same resource are coordinated between applications running on different nodes.

Summary

- Parallel databases improve processing and Input/Output speeds by using multiple CPUs and disks in parallel.
- Parallel database architectures can be broadly classified into three categories: shared memory, shared disk, and shared nothing.
- Individual relational operations (e.g., sort, join, aggregation) can be executed in parallel. Different queries can be run in parallel with each other. Concurrency control takes care of conflicts.
- I/O parallelism reduces the time required to retrieve relations from disk by partitioning the relations on multiple disks.
- Horizontal partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk.
- Partitioning techniques: Round-robin partitioning, Hash partitioning, Range partitioning
- Inter-query parallelism: Queries/transactions execute in parallel with one another.
- Intra-query parallelism: Execution of a single query in parallel on multiple processors/disks; important for speeding up long-running queries.
- Two complementary forms of Intra-query Parallelism: 1. Intra-operational Parallelism – parallelize the execution of each individual operation in the query. 2. Inter-operation Parallelism – execute the different operations in a query expression in parallel.
- The key elements of parallel database processing: Speed-up, Scale-up, Synchronisation, Locking
- Speed-up is the property of running a given task in less time by increasing the degree of parallelism (more number of hardware).
- Scale-up is the ability of handling larger tasks by increasing the degree of parallelism (providing more resources) in the same time period as the original system.
- Synchronisation is the coordination of concurrent tasks. *Locking* is a method of synchronising concurrent tasks.

Check Your Understanding

1. DBMS running across multiple processors is called as _____.
   (a) Distributed Systems (b) Parallel Systems
   (c) Centralized Systems (d) None of the above
2. Shared-memory architecture is also known as _____.
   (a) MPP (b) SMP
   (c) Both (a) and (b) (d) None of the above
3. A global locking system is required in _____.
   (a) Shared-disk architecture (b) Shared-nothing architecture
   (c) Shared-memory architecture (d) None of the above
4. Parallelism in which the relations are partitioned on multiple disks to reduce the retrieval time of relations from disk is called as _____.
   (a) I/O parallelism (b) Inter-operation parallelism
   (c) Shared-memory architecture (d) None of the above
5. Hash partitioning prevents _____.
   (a) Skewing (b) Loss of data
   (c) Speed-up (d) None of the above
6. Synchronisation is the coordination of _____.
   (a) Concurrent tasks (b) Serial tasks
   (c) Large tasks (d) None of the above
7. The architecture that is easy to load-balance is _____.
   (a) Shared-disk architecture (b) Shared-nothing architecture
   (c) Shared-memory architecture (d) None of the above
8. A method of synchronoising concurrent tasks is _____.
   (a) Synchronisation (b) Locking
   (c) Speed-up (d) None of the above
9. Which of the following is not a parallel database architecture ?
   (a) Shared memory (b) Shared processor
   (c) Shared Disk (d) Shared nothing
10. Range partitioning works best for_____.
    (a) Large databases (b) Small databases
    (c) Databases with less attributes (d) Databases with less range tuples

ANSWER KEY

1. (b)	2. (b)	3. (a)	4. (c)	5. (a)
6. (a)	7. (a)	8. (b)	9. (b)	10. (d)

Practice Questions

Q.I: Answer the following questions in short.

1. List a few relational operators that could be run in parallel.
2. Which are key elements of parallel data processing?
3. What is parallelism?
4. What are speed-up and scale-up key elements?
5. Which are categories of parallel database architecture?

Q.II: Answer the following questions.

1. What is parallel database? Explain with its advantages and disadvantages.
2. Explain in detail parallel database architecture.
3. Write in detail I/O parallelism.
4. What are Inter-query and Intra-query parallelism?
5. Describe Inter-operational and Intra-operational parallelism.

Q.III: Write a short note on:

1. Hash Partitioning
2. Range Partitioning
3. Round-robin Partitioning
4. Shared-disk architecture
5. Shared-nothing architecture

❖❖❖

5... Distributed Databases

Objectives...

- To introduce concept of Distributed Database System.
- To learn about Homogeneous and Heterogeneous Databases.
- To know about Distributed data storage (Fragmentation and Replication) and Distributed transactions.
- To get information of Concurrency controls schemes and commit protocols in DDBMS.

5.1 INTRODUCTION TO DISTRIBUTED DATABASE SYSTEM

- For the proper functioning of any organization, there's a need for a well-maintained database. In the recent past, databases used to be centralized in nature. However, with the increase in globalization, organizations tend to be diversified across the globe. They may choose to distribute data over local servers instead of a central database. Thus, the concept of Distributed Databases has arrived.
- A Distributed Database Management System (DDBMS) is a set of multiple, logically interrelated databases distributed over a network. They provide a mechanism that makes the distribution of data transparent to users.
- Users access the distributed database via applications, which 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 a DDBMS to have at least one global application.
- DDBMS is widely used in data warehousing, where huge volumes of data are processed and accessed by numerous users or database clients at the same time. This database system is used to manage data in networks, maintain confidentiality and handle data integrity.

5.1.1 Characteristics of DDBMS

- A DDBMS has the following characteristics:
  - A collection of logically related shared data.
  - The data is split into a number of fragments.
  - Fragments may be replicated.
  - Fragments/replicas are allocated to sites.
  - The sites are linked by a communications network.
  - The data at each site is under the control of a DBMS.
  - The DBMS at each site can handle local applications, autonomously.
  - Each DBMS participates in at least one global application.

5.1.2 Advantages and Disadvantages

Advantages:

- Following are advantages of DDBMS:
  1. **Sharing data:** Users at one site able to access the data residing at some other sites.
  2. **Autonomy:** Each site is able to retain a degree of control over data stored locally.
  3. **Higher system availability through redundancy:** Data can be replicated at remote sites and systems can function even if a site fails.

Disadvantages:

1. **Added complexity:** Required to ensure proper coordination among sites.
2. **Cost:** Extra Software development cost. This database is more expensive as it is complex and hence, difficult to maintain.
3. **Security:** Greater potential for bugs. The probability of security breaks increases when data are located at multiple sites. The responsibility of data management will be shared by different people at several sites.
4. **Processing overhead:** Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
5. **Lack of standard:** Although it provides effective communication and data sharing, still there are no standard rules and protocols to convert a centralized DBMS to a large Distributed DBMS. Thus, lack of standards decreases the potential of Distributed DBMS.

5.2 HOMOGENEOUS AND HETEROGENEOUS DATABASES

- Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments, each with further sub-divisions.

1. Homogeneous distributed databases:

- Homogeneous distributed database system is a network of two or more databases (With the same type of DBMS software) which can be stored on one or more machines.
- So, in this system data can be accessed and modified simultaneously on several databases in the network. Homogeneous distributed systems are easy to handle.

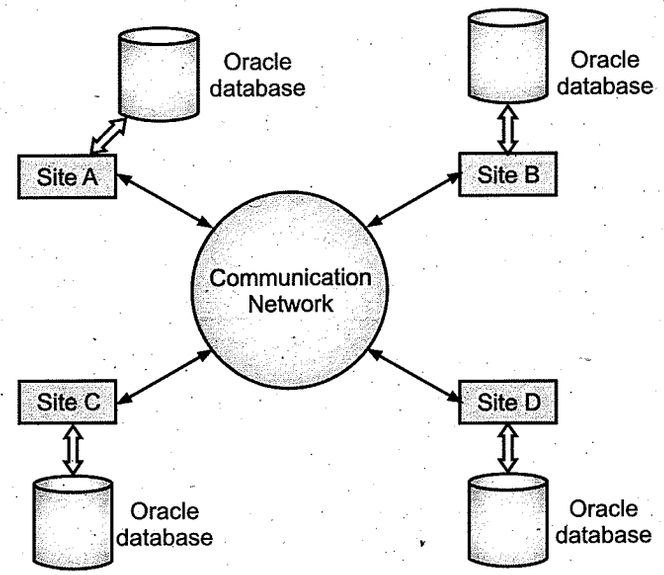


Fig. 5.1: Homogeneous DDBMS with same database (Oracle)

- **Example:** Consider that we have three departments using Oracle-9i for DBMS. If some changes are made in one department then it would update the other department also.

Types of Homogeneous Distributed Database:

- There are two types of homogeneous distributed database :
  - **Autonomous:** Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
  - **Non-autonomous:** Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

2. Heterogeneous Distributed Databases :

- Heterogeneous distributed database system is a network of two or more databases with different types of DBMS software, which can be stored on one or more machines.
- In this system, data can be accessible to several databases in the network with the help of generic connectivity (ODBC and JDBC).

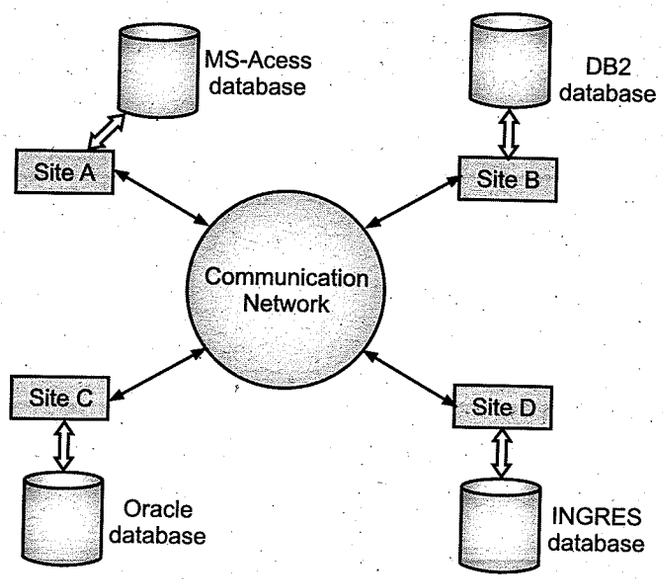


Fig. 5.2: Heterogeneous DDBMS with different DBMS

Types of Heterogeneous Distributed Databases:

1. **Federated:** The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
2. **Un-federated:** The database systems employ a central coordinating module through which the databases are accessed.

Difference between Homogeneous and Heterogeneous Distributed databases:

Homogeneous Distributed databases:

- Same software/schema on all sites, data may be partitioned among sites.
- Goal: provide a view of a single database, hiding details of distribution.

Heterogeneous Distributed databases:

- Different software/schema on different sites.
- Goal: integrate existing databases to provide useful functionality.

5.3 DISTRIBUTED DATA STORAGE (FRAGMENTATION AND REPLICATION)

- In this section, you will be introduced to distributed database design issues. These include data fragmentation, data replication and data allocation.

1. **Data Fragmentation:**

Data fragmentation is a technique used to break up objects. In designing a distributed database, you must decide which portion of the database is to be stored where. The database may be broken up into logical units called *fragments*. The simplest logical units are the tables themselves. Fragmentation information is stored in a distributed data catalogue which the processing computer uses to process a user's request.

Types of Data Fragmentation:

(i) **Horizontal fragmentation:** A horizontal fragment of a table is a subset of rows in it. So horizontal fragmentation divides a table 'horizontally' by selecting the relevant rows and these fragments can be assigned to different sides in the distributed system (for example, Tilak Road branch gets the fragment where myTable.branch = ' Tilak Road '). Each horizontal fragment may have a different number of rows, but each fragment must have the same attributes.

(ii) **Vertical fragmentation:** A vertical fragment of a table keeps only certain attributes of it. It divides a table vertically by columns. It is necessary to include the primary key of the table in each vertical fragment so that the full table can be reconstructed if needed. Each vertical fragment must have the same number of rows, but can have different attributes depending on the key.

(iii) **Mixed fragmentation:** In a mixed fragmentation, each fragment can be specified by a SELECT-PROJECT combination of operations. In this case, the original table can be reconstructed by applying union and natural join operations in the appropriate order.

In other words, this type of fragmentation is a two-step process. First, horizontal fragmentation is done to obtain the necessary rows and then vertical fragmentation is done to divide the attributes among the rows.

2. **Data Allocation:**

- Data allocation is a process of deciding where to store the data. It also involves a decision as to which data is stored at what location. Data allocation can be centralised, partitioned or replicated.
  - **Centralised:** The entire database is stored at one site. No distribution occurs.
  - **Partitioned:** The database is divided into several fragments that are stored at several sites.
  - **Replicated:** Copies of one or more database fragments are stored at several sites.

3. **Data Replication:**

- Data replication is the design process of deciding which fragments will be replicated. A copy of each fragment can be maintained at several sites, thus enhancing data availability and response time. Replicated data is subject to a mutual consistency rule. This rule requires that all copies of the data fragments must be identical and to ensure data consistency among all of the replications.
- Although data replication is beneficial in terms of availability and response times, the maintenance of the replications can become complex.

- For example, if data is replicated over multiple sites, the DDBMS must decide which copy to access. For a query operation, the nearest copy is all that is required to satisfy a transaction. However, if the operation is an update, then all copies must be selected and updated to satisfy the mutual consistency rule.
- Data replication is particularly useful if the usage frequency of remote data is high and the database is fairly large. Another benefit of data replication is the possibility of restoring lost data at a particular site.
- A database can be either fully replicated, partially replicated or unreplicated.

1. **Full replication:** Stores multiple copies of each database fragment at multiple sites. Fully replicated databases can be impractical because of the amount of overhead imposed on the system.
2. **Partial replication:** Stores multiple copies of some database fragments at multiple sites. Most DDBMS can handle this type of replication very well.
3. **No replication:** Stores each database fragment at a single site. No duplication occurs.

5.4 DATA TRANSPARENCY

- The user of a database system should not be required to know either where the data are physically located or how the data can be accessed at the specified local site.
- This characteristic feature is called Data Transparency. It is of 3 types:

1. **Fragmentation Transparency:**

- Users are not required to know how a relation has been fragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites. This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

2. **Replication Transparency:**

- Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed. It enables users to query upon a table as if only a single copy of the table exists.

3. **Location Transparency:**

- Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally on the user's site. The fact that users are not required to know the physical location of the data. The address of the remote site(s) and the access mechanisms are completely hidden.
- The distributed database system should be able to find any data as long as the data identifier is supplied by the user transaction.

5.5 DISTRIBUTED TRANSACTIONS

5.5.1 Concept of Transaction

- A transaction is a sequence of actions on a database that forms a basic unit of reliable and consistent computing, and satisfies the ACID property. In a distributed database system (DDBS), transactions may be local or global.
- A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, assume the database configuration depicted in Fig. 5.3:

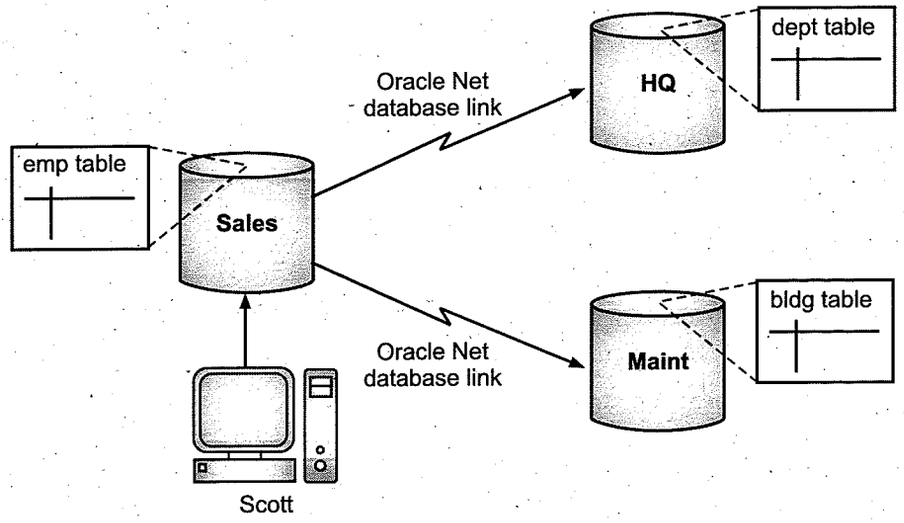


Fig. 5.3: Distributed System

- The following distributed transaction executed by **Scott** updates the local **Sales** database, the remote **HQ** database, and the remote **Maint** database:

```
UPDATE scott.dept@hq.us.acme.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.acme.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

5.5.2 Operations in Distributed Transaction

- There are two types of permissible operations in distributed transactions:
  1. DML and DDL Transactions.
  2. Transaction Control Statements.

1. DML and DDL Transactions:

- The following are the DML and DDL operations supported in a distributed transaction:
  - CREATE TABLE AS SELECT
  - DELETE
  - INSERT (default and direct load)
  - LOCK TABLE
  - SELECT
  - SELECT FOR UPDATE
- You can execute DML and DDL statements in parallel, and INSERT operation direct load statements serially, but note the following restrictions:
  - All remote operations must be SELECT statements.
  - These statements must not be clauses in another distributed transaction.
  - If the table referenced in the table_expression_clause of an INSERT, UPDATE, or DELETE statement is remote, then execution is serial rather than parallel.
  - You cannot perform remote operations after issuing parallel DML/DDL or direct load INSERT.
  - If the transaction begins using XA or OCI, it executes serially.
  - No loopback operations can be performed on the transaction originating the parallel operation. For example, you cannot reference a remote object that is actually a synonym for a local object.
  - If you perform a distributed operation other than a SELECT in the transaction, no DML is parallelized.

2. Transaction Control Statements:

- The following are the supported transaction control statements:
  - COMMIT
  - ROLLBACK
  - SAVEPOINT
- **COMMIT:** A COMMIT statement means that the changes made during a transaction to a particular database (or a set of databases) are permanent and become visible to other users.

- **ROLLBACK:** A ROLLBACK statement will allow you to cancel the modifications made during the current transaction.
- **SAVEPOINT:** This statement used to identify a point in a transaction to which you can later roll back.

5.5.3 Session Trees for Distributed Transactions

- When the statements in a distributed transaction are issued, the database defines a session tree of all nodes participating in the transaction. A **session tree** is a hierarchical model that describes the relationships among sessions and their roles. Fig. 5.4 illustrates a session tree:

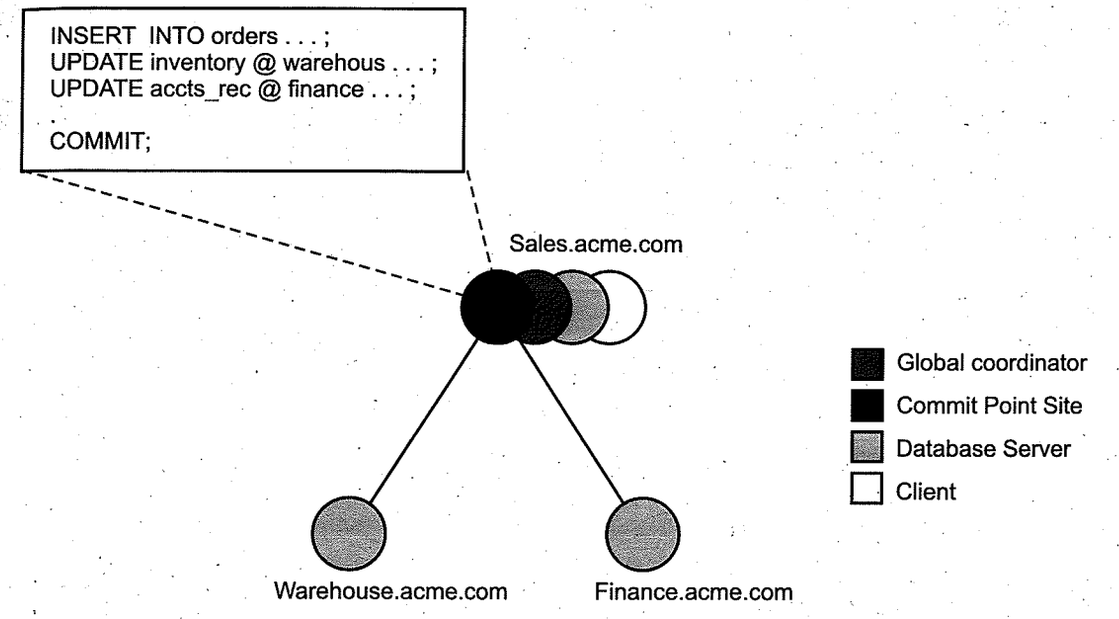


Fig. 5.4: A Session Tree

- All nodes participating in the session tree of a distributed transaction assume one or more of the following roles:

Table 5.1: Roles of Session Tree's Nodes

Role	Description
Client	A node that references information in a database belonging to a different node.
Database Server	A node that receives a request for information from another node.
Global Coordinator	The node that originates the distributed transaction.

Contd..:

Local Coordinator	A node that is forced to reference data on other nodes to complete its part of the transaction.
Commit Point Site	The node that commits or rolls back the transaction as instructed by the global coordinator.

- The role a node plays in a distributed transaction is determined by:
  - Whether the transaction is local or remote.
  - The commit point strength of the node ("Commit Point Site").
  - Whether all requested data is available at a node, or whether other nodes need to be referenced to complete the transaction.
  - Whether the node is read-only.

Let us see what is the role of each node of the session tree.

- **Clients:** A node acts as a client when it references information from a database on another node. The referenced node is a database server. In Fig.5.4, the node Sales is a client of the nodes that host the Warehouse and Finance databases.
- **Database Servers:** A database server is a node that hosts a database from which a client requests data. In Fig. 5.4, an application at the Sales node initiates a distributed transaction that accesses data from the Warehouse and Finance nodes. Therefore, *sales.acme.com* has the role of client node, and Warehouse and Finance are both database servers. In this example, Sales is a database server and a client because the application also modifies data in the Sales database.
- **Local Coordinators:** A node that must reference data on other nodes to complete its part in the distributed transaction is called a Local Coordinator. In Fig. 5.4, Sales is a local coordinator because it coordinates the nodes it directly references: Warehouse and Finance. The node Sales also happens to be the Global Coordinator because it coordinates all the nodes involved in the transaction.
- A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:
  - Receiving and relaying transaction status information to and from those nodes.
  - Passing queries to those nodes.
  - Receiving queries from those nodes and passing them on to other nodes.
  - Returning the results of queries to the nodes that initiated them.
- **Global Coordinator:** The node where the distributed transaction originates is called the Global Coordinator. The database application issuing the distributed transaction is directly connected to the node acting as the global coordinator. For example, in Fig. 5.4, the transaction issued at the node Sales references information from the database servers : Warehouse and Finance. Therefore, *sales.acme.com* is the global coordinator of this distributed transaction.

- The global coordinator becomes the parent or root of the session tree. The global coordinator performs the following operations during a distributed transaction:
  - Sends all of the distributed transaction SQL statements, remote procedure calls, and so forth to the directly referenced nodes, thus forming the session tree.
  - Instructs all directly referenced nodes other than the commit point site to prepare the transaction.
  - Instructs the commit point site to initiate the global commit of the transaction if all nodes prepare successfully.
  - Instructs all nodes to initiate a global rollback of the transaction if there is an abort response.
- **Commit Point Site:** The job of the commit point site is to initiate a commit or rollback operation as instructed by the global coordinator. The system administrator always designates one node to be the commit point sites in the session tree by assigning all nodes commit point strength. The node selected as commit point site should be the node that stores the most critical data.
- Fig. 5.5 illustrates an example of a distributed system, with Sales serving as the commit point site:

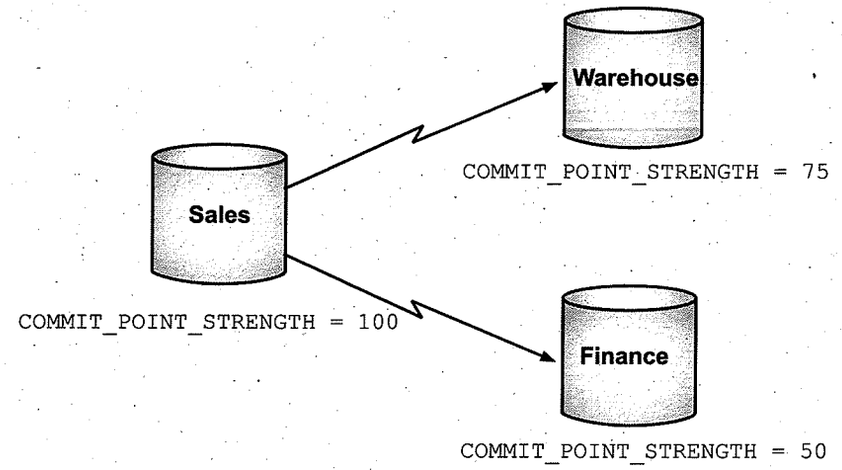


Fig. 5.5: Commit Point Site

- The commit point site is distinct from all other nodes involved in a distributed transaction in these ways:
  - The commit point site never enters the prepared state. Consequently, if the commit point site stores the most critical data, this data never remains in-doubt, even if a failure occurs. In failure situations, failed nodes remain in a prepared state, holding necessary locks on data until in-doubt transactions are resolved.
  - The commit point site commits before the other nodes involved in the transaction. In effect, the outcome of a distributed transaction at the commit point site

determines whether the transaction at all nodes is committed or rolled back: the other nodes follow the lead of the commit point site. The global coordinator ensures that all nodes complete the transaction in the same manner as the commit point site.

5.5.4 How a Distributed Transaction Commits?

- A distributed transaction is considered committed after all non-commit-point sites are prepared, and the transaction has been actually committed at the commit point site. The redo log at the commit point site is updated as soon as the distributed transaction is committed at this node.
- Because the commit point log contains a record of the commit, the transaction is considered committed even though some participating nodes may still be only in the prepared state and the transaction not yet actually committed at these nodes. In the same way, a distributed transaction is considered not committed if the commit has not been logged at the commit point site.

Commit Point Strength:

- Every database server must be assigned commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit. Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction and thus commits before all of the other nodes. This value is specified using the initialization parameter COMMIT_POINT_STRENGTH. This section explains how the database determines the commit point site.
- The commit point site, which is determined at the beginning of the prepare phase, is selected only from the nodes participating in the transaction. The following sequence of events occurs:
  1. Of the nodes directly referenced by the global coordinator, the database selects the node with the highest commit point strength as the commit point site.
  2. The initially-selected node determines if any of the nodes from which it has to obtain information for this transaction has higher commit point strength.
  3. Either the node with the highest commit point strength directly referenced in the transaction or one of its servers with higher commit point strength becomes the commit point site.
  4. After the final commit point site has been determined, the global coordinator sends *prepared* responses to all nodes participating in the transaction.
- Fig. 5.6 shows in a sample session tree the commit point strengths of each node (in parentheses) and shows the node chosen as the commit point site:

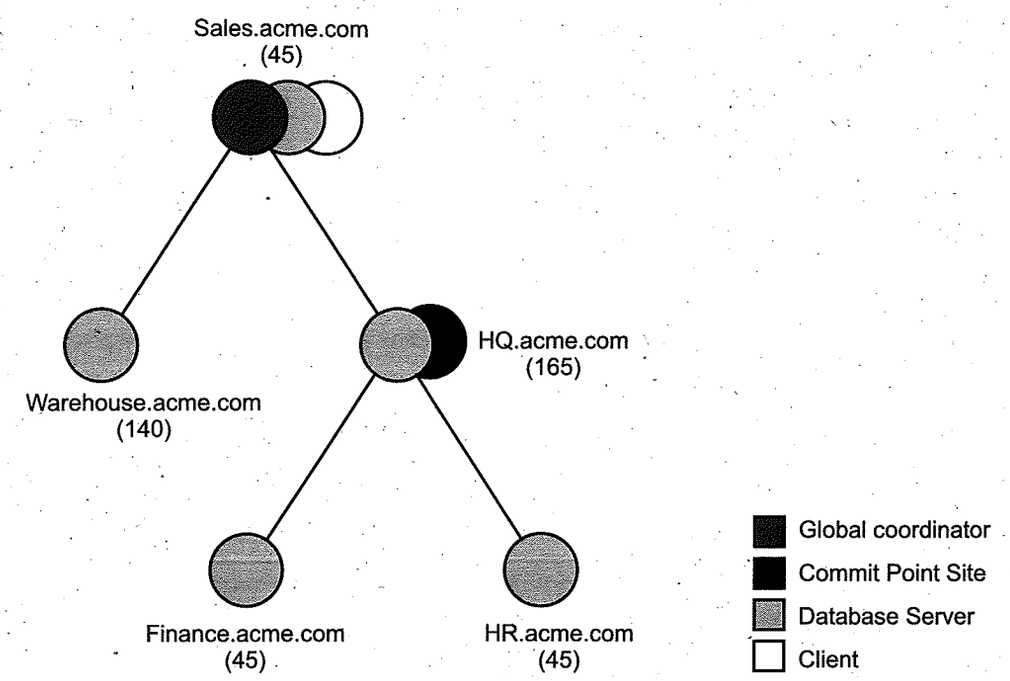


Fig. 5.6: Commit Point Strengths and Determination of the Commit Point Site

- When determining the commit point site, the following conditions are apply:
  - A read-only node cannot be the commit point site.
  - If multiple nodes directly referenced by the global coordinator have the same commit point strength, then the database designates one of these as the commit point site.
  - If a distributed transaction ends with a rollback, then *prepare* and *commit* phases are not needed. Consequently, the database never determines a commit point site. Instead, the global coordinator sends a ROLLBACK statement to all nodes and ends the processing of the distributed transaction.
- Fig. 5.6 illustrates, the commit point site and the global coordinator can be different nodes of the session tree. The commit point strength of each node is communicated to the coordinators when the initial connections are made. The coordinators retain the commit point strengths of each node they are in direct communication with so that commit point sites can be efficiently selected during two-phase commits. Therefore, it is not necessary for the commit point strength to be exchanged between a coordinator and a node each time a commit occurs.

5.6 CONCURRENCY CONTROL SCHEMES IN DDBMS

- Different techniques are provided by distributed DBMS for controlling the concurrency. These are :

1. **Locking Based Concurrency Control Protocols:** In order to decide whether a data item can perform read/write operations, a variable is connected to the data item, which is known as a lock. The possibility of using two transactions for locking a data item is identified by a lock compatibility matrix. The protocols used by these control systems may be either one-phase or two-phase locking protocols.
   (i) **One-phase Locking Protocol:** An item before it is used is locked by the transaction and once the item is used, it is released. Concurrency at its core is provided by this method.
   (ii) **Two-phase Locking Protocol:** The unlock operations are preceded by the first lock-release operation. A particular transaction is divided into two different phases. All the required locks are obtained by a transaction in the first phase and none of the locks are released. The first phase is also known as *growing* phase. All the locks are released by the transaction in the second phase and is known as *shrinking* phase. A transaction is said to be serializable when it takes up the concept of two-phase protocol. The two transactions which are conflicting in nature cannot be made parallel by this approach.
2. **Timestamp Concurrency Control Algorithms:** In order to coordinate with the concurrent access of a data item, the timestamp of a transaction is used. A timestamp is a unique identifier given by DBMS to a transaction that represents the transaction's start time.

- In accordance the age of the transactions, the serializable schedules are generated by the timestamp-based concurrency control techniques.
- Some of the algorithms associated with the timestamp based concurrency control are:
  - Basic timestamp ordering algorithm.
  - Conservative timestamp ordering algorithm.
  - Multiversion algorithm based upon timestamp ordering.
- Serializability is implemented by following some of the rules by timestamp. The rules are as follows:
  - **Access Rule:** When the same data is accessed by two different transactions pertaining to different operations, the transaction that is older is given the first chance. Hence the older transaction commits first and the younger transaction is made to wait.
  - **Late Transaction Rule:** The older transaction is not permitted to read or write the data item written by the younger transaction. The commitment of the older transaction is prevented after the younger transaction is committed.
  - **Younger Transaction Rule:** The data item that is written by the older transaction can be read or written by the younger transaction.

1. **Optimistic Concurrency Control Algorithm:** The rate of performance may be reduced by the process of validating each and every transaction. Hence the serializability test is performed when the transaction is about to commit. This enables to reduce the probability of aborting non serializable transactions. This technique is known as Optimistic Concurrency Control Algorithm.

- This algorithm is implemented by segregating the life cycle of a transaction into three different phases. They are:

1. **Execution Phase:** The data items are moved to the memory by the transactions and then they are operated.
2. **Validation Phase:** The changes to the database has to pass the test of serializability and this is ensured by the performs checks of the transaction.
3. **Commit Phase:** The data item after modification is written back in the memory.

- In the phase of validation, serializability is ensured by this algorithm by following three rules. They are:

Rule 1: Consider two transactions Ti and Tj, when the data items are being read by the Ti and are written by Tj, the commitment phase of Tjcannot be overlapped with the execution phase of Ti. In other words only once the execution is finished by Ti, Tj can start its commitment.

Rule 2: Consider two transactions Ti and Tj, if the data item is written by Ti and it is read by Tj, the execution phase of Tj cannot be overlapped by Ti commitment phase. Only once the Ti has committed, Tj can start executing.

Rule 3: Consider two transactions Ti and Tj, if Ti is writing the data item which Tj is also writing, then Ti's commit phase cannot overlap with Tj's commit phase. Tj can start to commit only after Ti has already committed.

How Concurrency is controlled in a Distributed System?

- The above concurrency techniques are implemented in a distributed database system in the following way:

1. **Distributed Two-phase Locking Algorithm:**

- In a distributed system, there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.
- Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types:
  (i) **Centralized two-phase locking:** In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.

(ii) **Primary copy two-phase locking:** In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.

(iii) **Distributed two-phase locking:** In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored at its local site. The location of the lock manager is based upon data distribution and replication.

2. Distributed Timestamp Concurrency Control:

- In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any site's local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises a combination of site ID and that site's clock reading.
- For implementing timestamp ordering algorithms, each site has a scheduler that maintains a separate queue for each transaction manager. During transaction, a transaction manager sends a lock request to the site's scheduler. The scheduler puts the request to the corresponding queue in increasing timestamp order. Requests are processed from the front of the queues in the order of their timestamps, i.e. the oldest first.

3. Conflict Graphs:

- Another method is to create conflict graphs. For this transaction classes are defined. A transaction class contains two set of data items called read set and write set.
- A transaction belongs to a particular class if the transaction's read set is a subset of the class' read set and the transaction's write set is a subset of the class' write set.
- In the read phase, each transaction issues its read requests for the data items in its read set. In the write phase, each transaction issues its write requests.
- A conflict graph is created for the classes to which active transactions belong. This contains a set of vertical, horizontal, and diagonal edges.
  - A vertical edge connects two nodes within a class and denotes conflicts within the class.
  - A horizontal edge connects two nodes across two classes and denotes a write-write conflict among different classes.
  - A diagonal edge connects two nodes across two classes and denotes a write-read or a read-write conflict among two classes.
- The conflict graphs are analyzed to determine whether two transactions within the same class or across two different classes can be run in parallel.

4. Distributed Optimistic Concurrency Control Algorithm:

- Distributed optimistic concurrency control algorithm extends optimistic concurrency control algorithm. For this extension, following two rules are applied :

Rule 1: According to this rule, a transaction must be validated locally at all sites when it executes. If a transaction is found to be invalid at any site, it is aborted. Local validation guarantees that the transaction maintains serializability at the sites where it has been executed. After a transaction passes local validation test, it is globally validated.

Rule 2: According to this rule, after a transaction passes local validation test, it should be globally validated. Global validation ensures that if two conflicting transactions run together at more than one site, they should commit in the same relative order at all the sites they run together. This may require a transaction to wait for the other conflicting transaction, after validation before commit. This requirement makes the algorithm less optimistic since a transaction may not be able to commit as soon as it is validated at a site.

5.7 COMMIT PROTOCOLS: 2-PHASE AND 3-PHASE COMMIT PROTOCOL

- Commit protocols are used to ensure atomicity across sites:
  - A transaction which executes at multiple sites must be either committed at all the sites, or aborted at all the sites.
  - Not acceptable to have a transaction committed at one site and aborted at another.
- Commit protocol is of two types: Two-phase Commit Protocol, Three-phase Commit Protocol.

5.7.1 Two-phase Commit Protocol

- The two-phase commit protocol (2PC) requires a Global Recovery Manager, or Coordinator, to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables). The two-phase commit (2 PC) protocol is widely used.

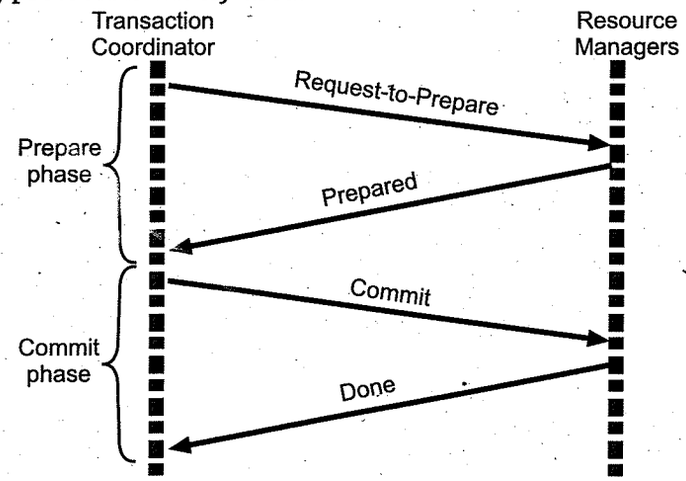


Fig. 5.7: The Two-phase Commit Protocol

- The steps performed in the two phases are as follows :

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site. When the controlling site has received "DONE" message from all slaves, it sends a "Prepare" message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a "Ready" message.
- A slave that does not want to commit sends a "Not Ready" message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received "Ready" message from all the slaves –
  - The controlling site sends a "Global Commit" message to the slaves.
  - The slaves apply the transaction and send a "Commit ACK" message to the controlling site.
  - When the controlling site receives "Commit ACK" message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first "Not Ready" message from any slave –
  - The controlling site sends a "Global Abort" message to the slaves.
  - The slaves abort the transaction and send an "Abort ACK" message to the controlling site.
  - When the controlling site receives "Abort ACK" message from all the slaves, it considers the transaction as aborted.

Drawbacks:

- The two-phase commit protocol has certain drawbacks that led to the development of the three-phase commit protocol.
- The biggest drawback of 2PC is that it is a blocking protocol. Failure of the coordinator blocks all participating sites, causing them to wait until the coordinator recovers. This can cause performance degradation, especially if participants are holding locks to shared resources.
- Another problematic situation is when both the coordinator and a participant that has committed crash together. In the two-phase commit protocol, a participant has no way to ensure that all participants got the commit message in the second phase. Hence once a decision to commit has been made by the coordinator in the first phase, participants will commit their transactions in the second phase independent of receipt of a global commit message by other participants.

- Thus, in the situation that both the coordinator and a committed participant crash together, the result of the transaction becomes uncertain or nondeterministic. Since the transaction has already been committed by one participant, it cannot be aborted on recovery by the coordinator. Also, the transaction cannot be optimistically committed on recovery since the original vote of the coordinator may have been to abort.

5.7.2 Three-phase (3PC) Commit Protocol

- Problems arises in 2PC protocol are solved by the three-phase commit (3PC) protocol. This protocol essentially divides the second commit phase into two sub-phases called **Prepare-to-Commit** and **Commit**.
- The **Prepare-to-Commit** phase is used to communicate the result of the vote phase to all participants. If all participants vote yes, then the coordinator instructs them to move into the prepare-to-commit state.
- The **Commit** sub-phase is identical to its two-phase counterpart. Now, if the coordinator crashes during this sub-phase, another participant can see the transaction through to completion. It can simply ask a crashed participant if it received a prepare-to-commit message. If it did not receive message then it safely assumes to abort the transaction. Thus the state of the protocol can be recovered irrespective of which participant crashes.
- Also, by limiting the time required for a transaction to commit or abort to a maximum time-out period, the protocol ensures that a transaction attempting to commit via 3PC releases locks on time-out.
- The main idea is to limit the wait time for participants who have committed and are waiting for a global commit or abort from the coordinator. When a participant receives a precommit message, it knows that the rest of the participants have voted to commit. If a precommit message has not been received, then the participant will abort and release all locks.
- In short, the steps in distributed three-phase commit are as follows :

Phase 1: Prepare Phase

- The steps are same as in distributed two-phase commit.

Phase 2: Prepare to Commit Phase

- The controlling site issues an "Enter Prepared State" broadcast message.
- The slave sites vote "OK" in response.

Phase 3: Commit / Abort Phase

- The steps are same as two-phase commit except that "Commit ACK"/"Abort ACK" message is not required.

Summary

- A Distributed Database Management System (DDBMS) is a set of multiple, logically interrelated databases distributed over a network. They provide a mechanism that makes the distribution of data transparent to users.
- In a homogeneous distributed database, all the sites use identical DBMS and operating systems.
- In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models.
- There are 2 ways in which data can be stored on different sites. These are: Replication and fragmentation.
- In replication, systems maintain copies of data. In fragmentation, the relations are fragmented (i.e., they are divided into smaller parts) and each of the fragments is stored in different sites where they are required.
- There are two main types of fragmentation: 1) Horizontal fragmentation 2) Vertical fragmentation.
- Distribution transparency allows users to perceive the database as a single, logical entity.
- A distributed transaction includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database.
- Commit protocol is of two types: Two phase commit protocol and Three phase commit protocol.
- The two-phase commit protocol breaks a database commit into two phases to ensure correctness and fault tolerance in a distributed database system.
- The three-phase commit (3PC) protocol is an extension of the two-phase commit protocol that avoids the blocking problem. This protocol avoids blocking by introducing an extra third phase where multiple sites are involved in the decision to commit

Check Your Understanding

1. _____ protocol is used to perform multiple transactions that execute on a different database.
   (a) commit (b) two-phase lock
   (c) two-phase commit (d) locking
2. A transaction that completes its execution successfully is said to be ____
   (a) committed (b) rolled back
   (c) partially committed (d) Aborted
3. The node where the distributed transaction originates is called the _____.
   (a) local coordinator (b) starting coordinator
   (c) global coordinator (d) originating node

4. A distributed database can use which of the following strategies?
   (a) Totally centralized at one location and accessed by many sites.
   (b) Partially or totally replicated across sites.
   (c) Partitioned into segments at different sites.
   (d) All of the above
5. Storing separate copy of the database at multiple locations is which of the following?
   (a) Data replication (b) Horizontal partitioning
   (c) Vertical Partitioning (d) Mixed partitioning
6. A distributed database is a collection of data which belong _____to the same system but are spread over the _____ of the network.
   (a) Logically, sites (b) Physically, sites
   (c) Database, DBMS (d) None of the above
7. Which of the following is/are the main goals of a distributed database?
   (a) Interconnection of database (b) Incremental growth
   (c) Reduced communication overhead (d) All of the above
8. Which of the following is a heterogeneous distributed database?
   (a) The same DBMS is used at each location and data are not distributed across all nodes.
   (b) The same DBMS is used at each location and data are distributed across all nodes.
   (c) A different DBMS is used at each location and data are not distributed across all nodes.
   (d) A different DBMS is used at each location and data are distributed across all nodes,
9. 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
10. A distributed transaction can be _____ if queries are issued at one or more nodes.
    (a) fully read-only (b) partially read-only
    (c) fully read-write (d) partially read-write

ANSWER KEY

1. (c)	2. (a)	3. (c)	4. (d)	5. (a)
6. (a)	7. (d)	8. (d)	9. (b)	10. (b)

Practice Questions

Q.I: Answers the following questions in short.

1. Which are types of DDBMS?
2. What is data transparency?
3. Which are states of transaction?
4. Define the data fragmentation?
5. Write difference between homogeneous and heterogeneous DDBMS.

Q.II: Answers the following questions.

4. What is DDBMS? Explain with its advantages and disadvantages.
5. Describe in detail the distributed transaction?
6. Explain two-phase and three-phase commit protocol.
7. Explain in detail concurrency control schemes?
8. Explain the term distribution transaction.

Q.III: Write a short note on:

1. Mixed fragmentation
2. Vertical fragmentation
3. Replication transparency
4. Transaction management
5. Three-phase commit protocol

❖❖❖

6... Object Oriented Databases and Applications

Objectives...

- To learn Object Oriented Database concepts & characteristics.
- To know about Database design for OODBMS.
- To get information of Spatial data and Spatial indexing.
- To study of Mobile database, temporal databases and Multimedia database.

6.1 INTRODUCTION

- Today's trend in programming languages is to utilize objects, thereby making OODBMS is ideal for Object Oriented programmers because they can develop the product, store them as objects, and can replicate or modify existing objects to make new objects within the OODBMS.
- An Object Oriented Database Management System (OODBMS) is the result of the combination of database techniques and object oriented concepts.
- An object oriented database is presently being used for various applications in areas such as e-commerce, engineering product data management; and special purpose databases in areas such as securities and medicine, telecommunications, and scientific areas such as high energy physics and molecular biology.
- Today many object oriented databases products are available like Objectivity/DB (developed by Objectivity, Inc.), ONTOS DB (developed by ONTOS, Inc.), VERSANT (developed by Versant Object Technology Corp.), ObjectStore (developed by Object Design, Inc.), GemStone (developed by Servio Corp.) and ObjectStore PSE Pro (developed by Object Design, Inc.).

6.2 OVERVIEW OF OBJECT-ORIENTED DATABASE CONCEPTS & CHARACTERISTICS

6.2.1 Concept of OODBMS

- OODBMS stands for Object Oriented Database Management System. It is a DBMS where data is represented in the form of objects, as used in object-oriented programming.
- OODB implements object-oriented concepts such as classes of objects, object identity, polymorphism, encapsulation, and inheritance. An object-oriented database stores complex data as compared to the relational database.
- OODBMS generally uses class definitions and traditional OOP languages, such as C++ and Java to define, manipulate and access data. That is, an OODBMS is an extension to the in-memory objects. An OODBMS integrates database handling operations directly into the base OOP language. The idea is shown in Fig. 6.1.

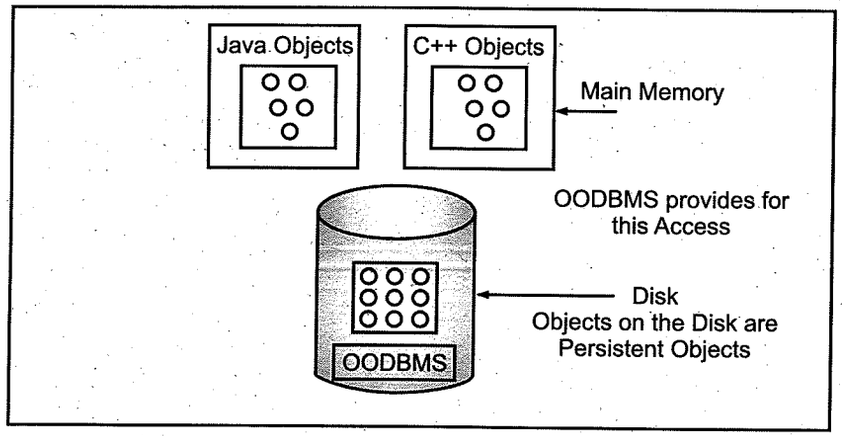


Fig. 6.1: Concept of OODBMS

6.2.2 Characteristics of OODBMS

The characteristics of object oriented database are listed below.

1. It keeps up a direct relation between real world and database objects as if objects do not lose their integrity and identity.
2. OODBs provide system generated object identifier for each object so that, an object can easily be identified and operated upon.
3. OODBs are extensible, which identifies new data types and the operations to be performed on them.
4. Provides encapsulation feature, in which the data representation and the methods implementation are hidden from external entities.
5. Also provides inheritance property, in which an object inherits the properties of other objects.

Object Oriented Methodologies:

There are certain object oriented methodologies are used in OODB. These are:

- **Class:** A class is assumed as a group of objects with the same or similar attributes and behavior.
- **Encapsulation:** It is the property that the attributes and methods of an object are hidden from the outside world. A published interface is used to access an object's methods.
- **Inheritance:** It is the property in which classes are arranged in a hierarchy. Each class assumes the attributes and methods of its ancestors. For example, class students are the ancestor of undergraduate students and post graduate students.
- **Polymorphism:** It allows several objects to represent the same message in different ways. In the object oriented database model, complex objects are modeled more naturally and easily.

6.2.3 Advantages and Disadvantages of OODBMS

- OODBMSs can provide appropriate solutions for many types of advanced database applications. However, there are also disadvantages.

Advantages:

1. **Enriched modeling capabilities :** The object-oriented data model allows the 'real world' to be modelled more closely. The object which encapsulates both state and behavior, is a more natural and realistic representation of real-world objects. An object can store all the relationships it has with other objects, including many-to-many relationships, and objects can be formed into complex objects that the traditional data models cannot cope with easily.
2. **Extensibility:** OODBMSs allow new data types to be built from existing types. The ability to factor out common properties of several classes and form them into a super-class that can be shared with sub-classes can greatly reduce redundancy within system is regarded as one of the main advantages of object orientation. Further, the reusability of classes promotes faster development and easier maintenance of the database and its applications.
3. **Capable of handling a large variety of data types:** Unlike traditional databases (such as hierarchical, network or relational), the object oriented database are capable of storing different types of data, for example, pictures, voice video, including text, numbers and so on.
4. **Removal of impedance mismatch:** A single language interface between the Data Manipulation Language (DML) and the programming language overcomes the impedance mismatch. This eliminates many of the efficiencies that occur in mapping a declarative language such as SQL to an imperative 'language such as 'C'. Most OODBMSs provide a DML that is computationally complete compared with SQL, the standard language of RDBMSs.

5. **More expressive query language:** Navigational access from the object is the most common form of data access in an OODBMS.
6. **Support for schema evolution:** The tight coupling between data and applications in an OODBMS makes schema evolution more feasible.
7. **Support for long-duration transactions:** Current relational DBMSs apply serializability on concurrent transactions to maintain database consistency. OODBMSs use a different protocol to handle the types of long-duration transaction that are common in many advanced database application.
8. **Applicability to advanced database applications:** There are many areas where traditional DBMSs have not been particularly successful, such as, Computer-Aided Design (CAD), Computer-Aided Software Engineering (CASE), Office Information System (OIS), and Multimedia Systems. The enriched modelling capabilities of OODBMSs have made them suitable for these applications.
9. **Improved performance:** There have been a number of benchmarks that have suggested OODBMSs provide significant performance improvements over relational DBMSs.

Disadvantages :

- There are following disadvantages of OODBMSs:

1. **Lack of universal data model:** There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation. This .disadvantage is seen as a significant drawback, and is comparable to per-relational systems.
2. **Lack of experience:** In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems. OODBMSs are still very much geared towards the programmer, rather than the naïve end-user. Also there is a resistance to the acceptance of the technology. While the OODBMS is limited to a small niche market, this problem will continue to exist
3. **Lack of standards:** There is a general lack of standards of OODBMSs. We have already mentioned that there is not universally agreed data model. Similarly, there is no standard object-oriented query language.
4. **Competition:** Perhaps one of the most significant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products. These products have an established user base with significant experience available. SQL is an approved standard and the relational data model has a solid theoretical formation and relational products have many supporting tools to help both end-users and developers.
5. **Query optimization compromises encapsulations:** Query optimization requires. An understanding of the underlying implementation to access the database efficiently. However, this compromises the concept of incrustation.

6. **Locking at object level may impact performance:** Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.
7. **Complexity:** The increased functionality provided by the OODBMS (such as the illusion of a single-level storage model, pointer sizzling, long-duration transactions, version management, and schema evolution–makes the system more complex than that of traditional DBMSs. In complexity leads to products that are more expensive and more difficult to use.
8. **Lack of support for views:** At present, most OODBMSs do not provide a view mechanism, which, as we have seen previously, provides many advantages such as data independence, security, reduced complexity, and customization.
9. **Lack of support for security:** At present, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes.

6.3 DATABASE DESIGN FOR OODBMS – OBJECTS, OIDS AND REFERENCE TYPE

- The OODBMS is based on three major components, namely: Object structure, Object classes, and Object identity.

6.3.1 Object

- An object is an abstract representation of the real world entity which has a unique identity, embedded properties, and the ability to interact with other objects and itself.

Characteristics of Object:

- Some important characteristics of an object are:

1. **Object name:** The name is used to refer different objects in the program.
2. **Object identifier:** This is the system generated identifier which is assigned, when a new object is created.
3. **Structure of object:** Structure defines, how the object is constructed using constructor.
   - In object oriented database, the state of complex object can be constructed from other objects by using certain type of constructor.
   - The formal way of representing objects as (i,c,v) where 'i' is object identifier, 'c' is type constructor and 'v' is current value of an object.
4. **Transient object:** In OOPL, objects which are present only at the time of execution are called a transient object. For example: Variables in OOPL.
5. **Persistent objects:** An object which exists even after the program is completely executed (or terminated), is called as persistent objects. Object-oriented databases can store objects in secondary memory.

Attributes:

- The attributes are the characteristics used to describe objects. Attributes are also known as instance variables. When attributes are assigned values at a given time, it is assumed that the object is in a given state at that time.

Type of Attributes:

- The three types of attributes are as follows:
  1. **Simple attributes:** Attributes can be of primitive data type such as, integer, string, real etc. which can take literal value. For example: 'ID' is simple attribute and value is 07.
  2. **Complex attributes:** Attributes which consist of collections or reference of other multiple objects are called as complex attributes. For example: Collection of Employees consists of many employee names.
  3. **Reference attributes:** Attributes that represent a relationship between objects and consist of value or collection of values are called as reference attributes. For example: Manager is reference of staff object.

6.3.2 Object Identity (OID)

- The identity is an external identifier. The object ID(OID) maintained for each object. The Object ID is assigned by the system when the object is created, and cannot be changed. It is unlike the relational database, for example, where a data value stored within the object is used to identify the object. In OODBMS OID are variable name or pointer.

Properties of OID:

1. **Uniqueness:** OID cannot be same to every object in the system and it is generated automatically by the system.
2. **Invariant:** OID cannot be changed throughout its entire lifetime.
3. **Invisible:** OID is not visible to user.

6.3.3 Reference Type

- OIDs are best used only for references to system tables.
- In SQL:2003, reference types can be used to define relationships between row types and uniquely identify a row within a table.
- Reference type value can be stored in one table and used as a direct reference to a specific row in some base table defined to be of this type (similar to pointer type in 'C'/C++).
- In this way, reference type provides similar functionality as OID of OODBMSs.
- Thus, references allow a row to be shared among multiple tables, and enable users to replace complex join definitions in queries with much simpler path expressions.
- References also give optimizer alternative way to navigate data instead of using value-based joins.

Example:

- Object-oriented languages provide the ability to refer to objects. An attribute of a type can be a reference to an object of a specified type.
- For example, in SQL:1999 we can define a type Department with a field name and a field head which is a reference to the type Person, and a table departments of type Department, as follows:

```
Create type Department (
name varchar(20),
head ref(Person) scope people
)
Create table departments of Department
```

- Here, the reference is restricted to tuples of the table people. The restriction of the scope of a reference to tuples of a table is mandatory in SQL:1999, and it makes references behave like foreign keys.
- We can omit the declaration scope people from the type declaration and instead make an addition to the create table statement:

```
Create table departments of Department
(head with options scope people)
```

- In order to initialize a reference attribute, we need to get the identifier of the tuple that is to be referenced. We can get the identifier value of a tuple by means of a query. Thus, to create a tuple with the reference value, we may first create the tuple with a null reference and then set the reference separately:

```
Insert into departments
   Values ('CS', null)
Update depatments
   Set head = (select ref(p)
               From people as p
               Where name = 'John')
   Where name = 'CS'
```

- This syntax for accessing the identifier of a tuple is based on the Oracle syntax. SQL:1999 adopts a different approach, one where the referenced table must have an attribute that stores the identifier of the tuple. We declare this attribute, called the self-referential attribute, by adding a ref is clause to the create table statement:

```
Create table people of Person
ref is oid system generated
```

- Here, `oid` is an attribute name, not a keyword. The subquery above would then use `select p.oid` instead of `select ref(p)`.
- An alternative to system-generated identifiers is to allow users to generate identifiers. The type of the self-referential attribute must be specified as part of the

type definition of the referenced table, and the table definition must specify that the reference is user generated:

```
Create type Person
    (name varchar(20),
    address varchar(20))
    ref using varchar(20)
Create table people of Person
    ref is old user generated
```

- When inserting a tuple in people, we must provide a value for the identifier.

```
Insert into people values
    ('01284567', 'John', '23 Coyote Run')
```

- No other tuple for people or its supertables or subtables can have the same identifier. We can then use the identifier value when inserting a tuple into departments, without the need for a separate query to retrieve the identifier.

```
Insert into departments
    values('CS', '01284567')
```

- It is even possible to use an existing primary key value as the identifier, by including the **ref from** clause in the type definition:

```
Create type Person
    (name varchar(20) primary key,
    address varchar(20))
    ref from(name)
Create table people of Person
    ref is old derived
```

- Note that, the table definition must specify that the reference is derived, and must still specify a self-referential attribute name. When inserting a tuple for departments, we can then use:

```
Insert into departments
    values ('CS', 'John')
```

6.4 SPATIAL DATA & SPATIAL INDEXING

- A spatial database is optimized to store and query data representing objects. These are the objects which are defined in a geometric space.
- In other words, it includes objects that have a SPATIAL location (and extent). A chief category of spatial data is geospatial data - derived from the geography of our earth.

Characteristics of Spatial Database:

- A spatial database system has the following characteristics:
  - It is a database system.
  - It offers spatial data types (SDTs) in its data model and query language.
  - It supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join.

6.4.1 Spatial Data

- Spatial data is associated with geographic locations such as cities, towns, etc.
- There are three general spatial data types that are typically used to represent geographic reality. We will focus on two dimensional types. The three types are: **point, line, and region**. Furthermore, they are differentiated into two classes: **Simple** and **Complex**.

Simplex class data types:

- A simple point is a single point in euclidean space: (x,y)
- A simple line is a connected, one dimensional object in euclidean space with two end points.
- A simple region is a single, simple, minimal cycle separating the interior of the region from the exterior of the region (A Jordan curve).

Fig. 6.2: Simplex class data types: Point, Line, Region

Complex class data types:

- A complex point, sometimes called a multi-point, is a collection of non-overlapping, simple points.
- A complex line is a collection of disjoint simple lines.
- A complex region is a collection of faces and holes. A face is a simple region, possibly containing holes. A hole fits the definition of a simple region, but the interior of a region lies on the exterior of a hole, and the exterior of a region lies on the interior of a hole. Faces can contain multiple holes, and multiple faces can occur within a hole. Faces and holes can touch along their boundaries at a finite number of discrete points.

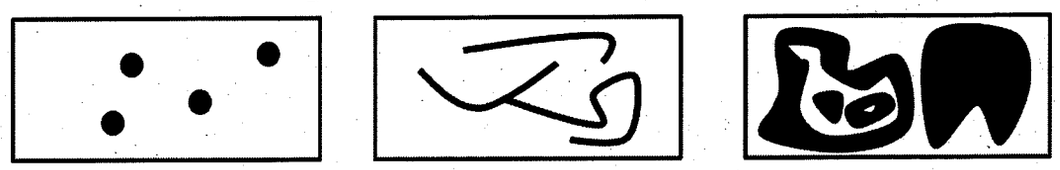


Fig. 6.3: Complex class data types: Point, Line, Region

- In mathematical terms, all data types are considered to be regular open point sets. This has desirable properties for the semantics of operations.
- All spatial objects have three components: interior, exterior and boundary.
  - Points have only a boundary and an exterior.
  - Lines have a boundary defined as the end points of the line. The interior is all parts of the line not at the end points. The exterior is everything else (the rest of the embedding space).

- Regions have a boundary that separates the interior and exterior of the region.

Integration of Spatial Types into Databases:

- The mathematical definitions of spatial types are based on point set topology. Types are defined by using infinite point sets, not directly representable in computers.

Examples:

- A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information. The location of cities, roads, and political boundaries that exist on the surface of the Earth is projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.
- The data that indicates the Earth location (latitude and longitude, or height and depth) of these rendered objects is the spatial data. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A GIS is often used to store, retrieve, and render this Earth-relative spatial data.
- Other types of spatial data that can be stored using the spatial option, besides GIS data include data from Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM) systems. Instead of operating on objects on a geographic scale, CAD/CAM systems work on a smaller scale such as for an automobile engine or printed circuit boards.
- The differences among these three systems are only in the scale of the data, not its complexity. They might all actually involve the same number of data points.
- On a geographic scale, the location of a bridge can vary by a few tenths of an inch without causing any noticeable problems to the road builders. Whereas, if the diameter of an engine's pistons are off by a few tenths of an inch, the engine will not run.
- A printed circuit board is likely to have many thousands of objects etched on its surface that are no bigger than the smallest detail shown on a road builder's blueprints.
- These applications all store, retrieve, update, or query some collection of features that have both non-spatial and spatial attributes.
- The spatial attribute is a coordinate geometry or vector-based representation of the shape of the feature. The spatial attribute, referred to as the geometry, is an ordered sequence of vertices that are connected by straight line segments or circular arcs. The semantics of the geometry are determined by its type, which may be one of point, line-string, or polygon.
- Examples of non-spatial attributes are name, soil_type, landuse_classification, and part_number.

6.4.2 Spatial Index

- A Spatial Index is a data-structure designed to enable fast access to spatial data. It is a common technique used by spatial databases.
- Common spatial index methods include: Geohash, HHCode, Grid (spatial index), Z-order (curve), Quadtree, Octree, UB-tree, R-tree etc.

6.4.2.1 Spatial Index Operations

- A variety of spatial operations needs the support from spatial index for efficient processing:
  - **Range query:** Finding objects containing a given point (point query) or overlapping with an area of interest (window query).
  - **Spatial join:** Finding pairs of objects that interact spatially with each other. Intersection, adjacency, and containment are common examples of spatial predicates to perform spatial joins.
  - **K-Nearest Neighbor (KNN):** Finding the nearest K spatial objects in a defined neighborhood of a target object.

Example:

- Here is a simple test executing the SELECT statement with and without a spatial index, finding corresponding features from three different datasets by a given point. Both SELECT operations yield the same results, however, the execution time increases when the data volume gets larger. Without indexing, any search for a feature would require a "sequential scan" of every record in the database, resulting in a much longer processing time.

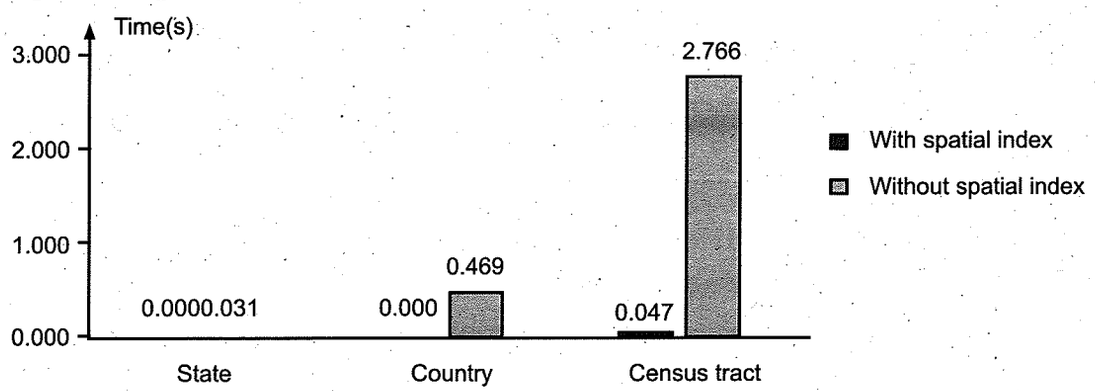


Fig. 6.4: Range query cost by a given point

(*In above fig, the point query test in Mysql51719 using 2017 TIGER national geodatasets. Time is measured in milliseconds which is the precision of Mysql*)

- In order to reduce the cost of calculating the complex shape of spatial object during the search traversal, we use approximations of the complex object geometries. The most commonly used approximation is the Minimum Bounding Rectangle, or MBR,

which is also called Minimum Bounding Box, or MBB. An MBR is a single rectangle that minimally encloses the geometry in 2D plane, as shown in Fig. 6.5 (a). It can also be extended to higher dimensions, such as a 3D MBR as in Fig. 6.5 (b).

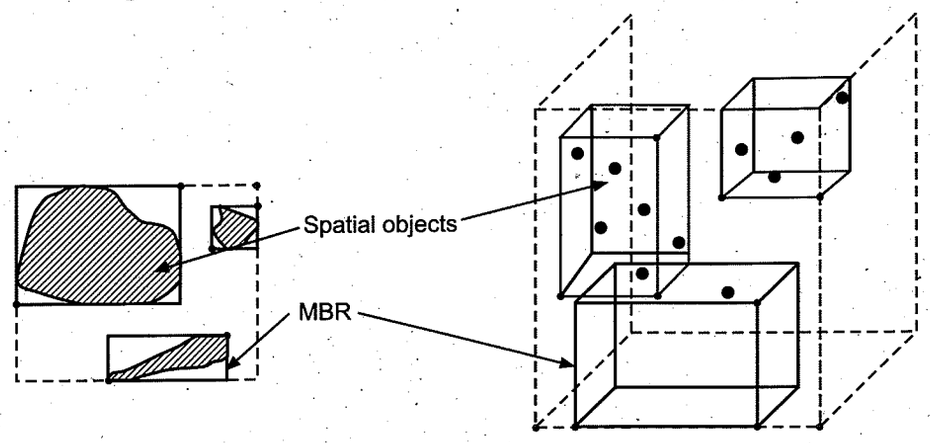


Fig. 6.5: Minimum bounding rectangles of (a) 2D objects and (b) 3D objects

- In a 2D plane, an MBR is defined by four coordinates, (x_{min}, y_{min}) and (x_{max}, y_{max}). These coordinates represent the following:
  - x_{min} is the x-coordinate of the lower-left corner of the bounding box.
  - y_{min} is the y-coordinate of the lower-left corner of the bounding box.
  - x_{max} is the x-coordinate of the upper-right corner of the bounding box.
  - y_{max} is the y-coordinate of the upper-right corner of the bounding box.
- When performing a spatial operation on a collection of indexed spatial objects, we first use MBR instead of the shape of spatial object to test the relation. By filtering spatial objects outside the MBR, time consuming on spatial predicates will reduce significantly. Then, in a second step, we use the actual shape in the subset of first step to test the spatial relation with the target object.

6.4.2.2 Spatial Index in Different Databases

- Different data sources use different data structures and access methods. Here we list two well-known spatial indices as well as databases that use them. The categorization system proposed by Riguax et al. (2002) is employed here for illustration:

i. **Space-driven structures:** These data structures are based on partitioning of the embedding 2D space into cells (or grids), mapping MBRs to the cells according to some spatial relationship (overlap or intersect).

   Commercial databases like IBM DB2 and Microsoft SQL Server use these methods. Besides, Esri also implement its spatial data access method on geodatabases using space-driven structures.

ii. **Data-driven structures:** These data structures are directly organized by partition of the collection of spatial objects. Data Objects are grouped using MBRs adapting to their distribution in the embedding space.

   Commercial databases like Oracle and Informix as well as some open-source databases, PostGIS and MySQL, use these data structures.

6.4.2.3 Space-driven Structures

- Space-driven structures of spatial objects are instinctively using partitioning and mapping strategies to decompose 2D plane into a list of cells, and can be used in spatial extensions with B+-tree, which is dynamic and efficient in memory space and query time.
- Although some of the indices can handle with arbitrary dimensions, we here describe four common partitioning structures in 2D case.

1. **Fixed grid index:**

- Fixed grid index is an $n \times n$ array of equal-size cells. Each one is associated with a list of spatial objects which intersect or overlap with the cell. Fig. 6.6 depicts a fixed 4×4 gird indexing a collection of three spatial objects.

Fig. 6.6: An Example of a Fixed Grid Structure.

- To further decrease the redundancy, a hierarchical uniform decomposition of the plane can be enabled. The index-creation process decomposes the plane into an m-level grid hierarchy (Microsoft SQL Server uses 4 level and IBM DB2 uses 3 level). These levels are referred to as level 1 (the top level), level 2, level 3... level m. The decomposition is more complex when the level gets higher. As shown in Fig. 6.7, each cell at a higher level is decomposed into a 4x4 grid at the next level.
- These grid hierarchy cells are numbered in a linear fashion called *space-filling curves*. They are useful because it partially preserves proximity, that is, two cells close in 2D plane are likely to be close in the sequential order. There are different spatial filling curves and here we use the z-order curve Fig. 6.8 (a) as an example of spatial filling curves. The Hilbert curve in Fig. 6.8 (b) is also well known.

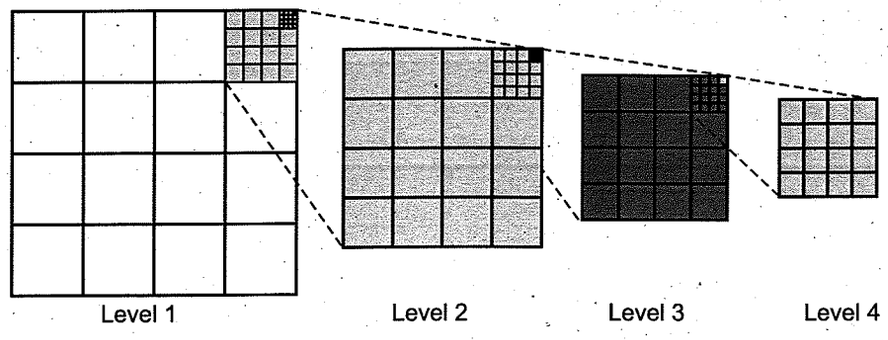


Fig. 6.7: A Four-level Fixed Grid Structure

- Z-order labels each cell like a complete quadtree and numbers each quadrant in binary format 00, 01, 10, 11. The number for a node is the concatenation of the quadrant numbers for each of its ancestors, starting at the root. For example, in Fig. 6.8 (a), "0" denotes a binary 00 for the top left, "1" denotes a binary 10 for the top right, and so forth. And the 2D plane gets further filling at a depth of 1, 2 and 3. This order is also called Morton code, which is used in raster storage.

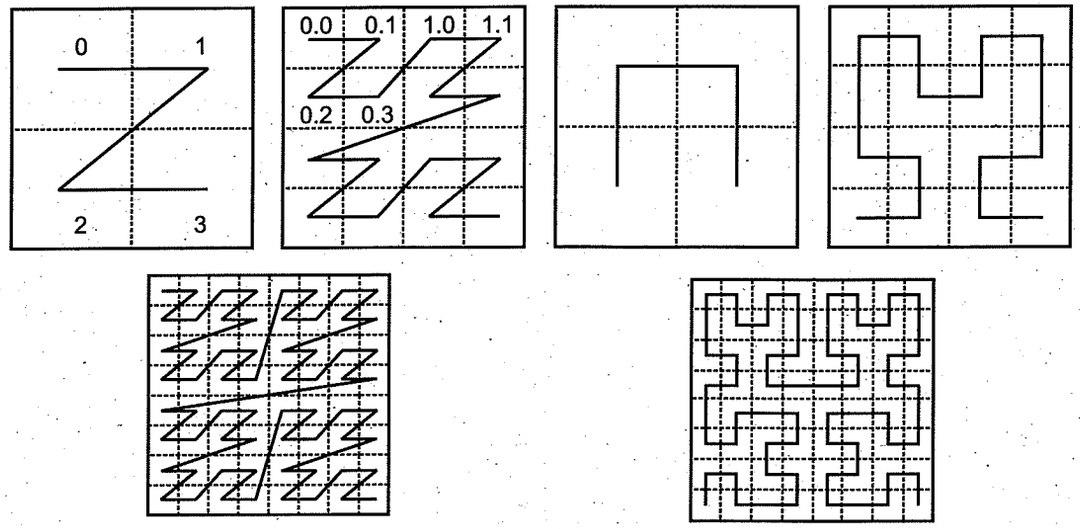


Fig 6.8: Two common space filing curves: (a) z-order and (b) Hilbert curve

2. Quadtree:

- Quadtree is a very popular spatial indexing technique. It is a specialized form of grid in which the resolution of the grid is varied according to the density of the spatial objects to be fitted.
- In a Quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area. Each node is either a leaf node which contains one or more indexed points, and no children, or it is an internal node, which has exactly four children, one for each quadrant obtained by dividing the

area covered in half along both axes. This is also how the name "quadtree" comes from.

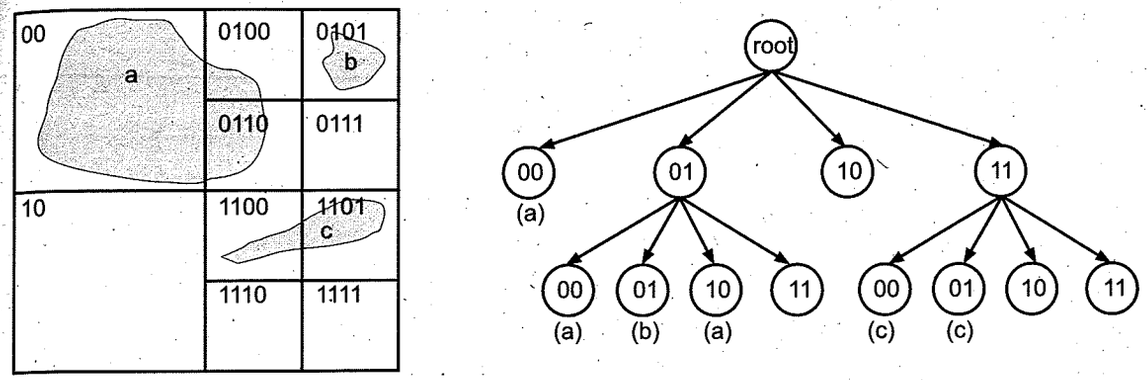


Fig. 6.9: A representation of a quadtree structure

3. KD-tree:

- The general idea behind KD-tree is that it is a binary tree; each of its nodes represents an axis-aligned hyper-rectangle as Fig. 6.10 shows. Each node specifies an axis and splits the set of points based on whether their coordinate along that axis is greater than or less than a particular value (Rigaux, 2012; Maneewongvatana, 1999), such as the coordinate median.

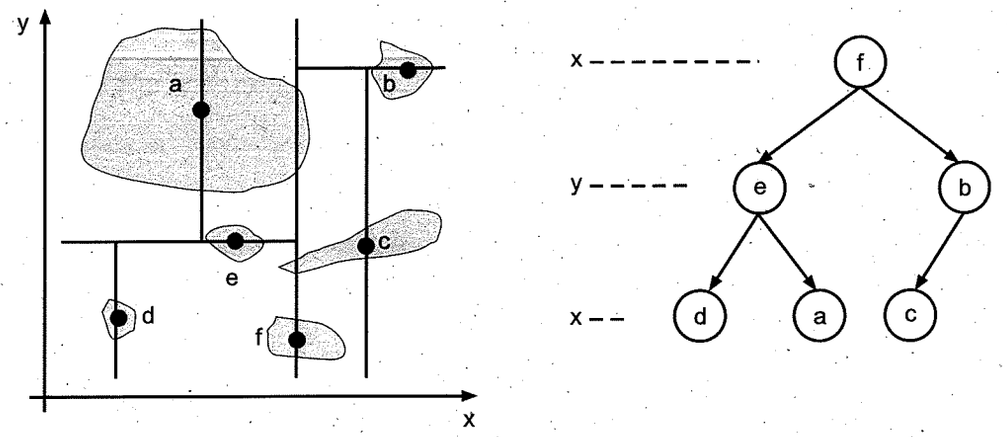


Fig. 6.10: A representation of a KD-tree structure

- The KD-tree can be used to index a set of k-dimensional points. Every non-leaf node divides the space into two parts by a hyper-plane in the specific dimension. Points in the left half-space are represented by the left subtree of that node and points falling to the right half-space are represented by the right subtree.
- The hyper-plane formation is in the following way. First, choose one dimension as split axis and a middle point, such as "x" axis and point "f" in Fig. 6.10. Then every point finds its position in the tree depending on its relative value to the middle point.

All points in the subtree with a smaller "x" value than the point "f" will appear in the left subtree and all points with larger "x" values will be in the right subtree. Continue to split axes and repeat the above steps to construct a KD-tree.

4. Geohash:

- Gustavo Niemeyer invented Geohash (link is external) in 2008 with the purpose of geocoding specific points as a short string to be used in web URLs.
- A Geohash is a binary string in which each character indicates alternating divisions of the global longitude-latitude rectangle. The process of partitioning spatial plane can be seen in following Fig. 6.11.
  - The first division splits the rectangle into two squares with a Geohash code of "0" and "1." Spatial objects locates at the left of the vertical division have a Geohash beginning with '0' and the ones in the right half have a Geohash beginning with "1" as its first prefix.
  - Then, data at each side is further split horizontally: objects that are above the line receive "0" and the ones below receive "1" as their second prefix.
  - The patterns continue to split until the desired resolution is reached. Every Geohash rectangle can be decomposed into four sub-hashes that are visited in a self-similar z-order curve.
- In order to make Geohash more useful for the web, the inventor assigned a plain text, base-32 and base-36 encoding for his web service to get a short web URL. The length of Geohash could range from 1 (in Fig. 6.11) to 12. A longer Geohash has a finer precision.

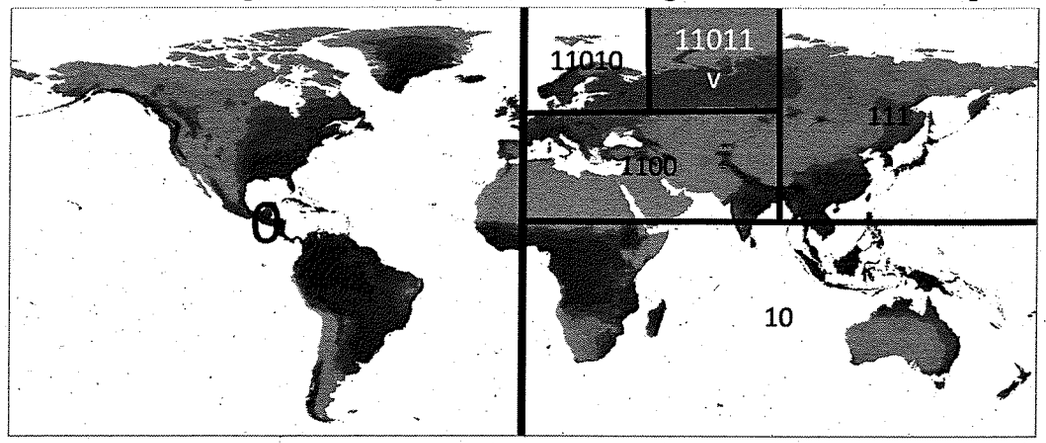


Fig. 6.11: An example of a Geohash (with a hash code "v")

6.4.2.4 Data-driven Structures

- The difference between space-driven structures and data-driven structures is that the latter use the idea of a spatial containment relationship instead of the order of the index. These structures adapt themselves to the MBR of the spatial objects and they are part of the R-tree family.

1. R-tree:

- For a layer of geometries, an R-tree index consists of a hierarchical index on the MBRs of the geometries in the layer (Guttman, 1984). This hierarchical structure is based on the heuristic optimization of the area of MBRs in each node in order to improve the access efficiency.

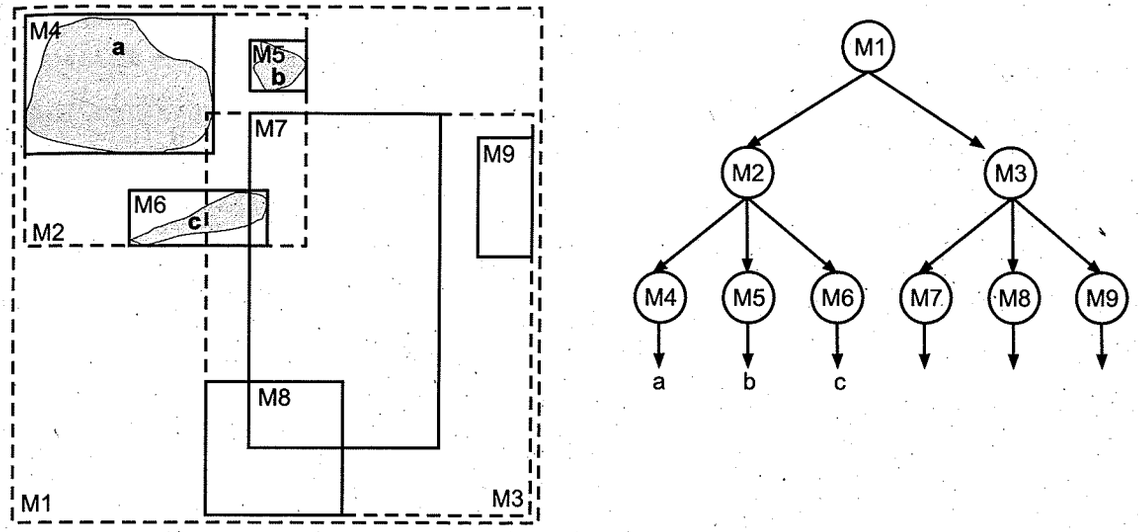


Fig. 6.12: R-Tree Hierarchical Index in Minimum Bounding Rectangles (MBRs)

- In Fig. 6.12 M4 through M9 are MBRs of spatial objects in a layer. They are the leaf nodes of the R-tree index, and contain minimum bounding rectangles of spatial objects, along with pointers to the spatial objects. M2 and M3 are parents of the leaf nodes. M1 is the root, containing all the MBRs. This R-tree has a depth of three.
- This kind of containment relationship could be extended to higher dimensions and has similar construction and query methods. Unlike the space-driven structures, spatial objects here only belong to one of the tree leaves and MBRs have overlap between nodes, such as M7 and M8 in Fig. 6.12.

2. Variants of R-Tree:

- Variants of the R-tree provide several improvements to the construction process (Beckmann et al., 1990), especially with regards to inserting and splitting of nodes. Essentially, these improvements aim at optimizing the following parameters: node overlapping, area covered by a node, and perimeter of a node's directory rectangle. The perimeter of a node's directory rectangle is representative of the shape of the MBR because, given a fixed area, the shape that minimizes the MBR perimeter is the square (in 2D) or cube (in 3D). There are no well-founded techniques to simultaneously optimize these three parameters. Thus, different variants bring different improvements with respect to the original R-tree. Three of the variants are described below.

1. The **R*-tree approach** assumes the split to be performed along one axis (say, x-axis), and explores all possible distributions of objects above or below the split line. Another characteristic is that R*-tree forces one to reinsert the same objects when structures are updated. This improves some degraded conditions.
2. **R-tree Packing** is also a well-known type of data structure. In this case, we use a pre-processing phase that sorts MBRs according to their location (for example, the *x* coordinate of the centroid) and then construct the nodes in a way that involves less overlap.
3. In the **R+-tree**, the MBR of nodes at a given level does not overlap. Objects that cross different MBRs appear twice or more. This is similar to some space-driven structures. As a result, a single path is followed from the root to a leaf for a point query.

6.5 MOBILE DATABASE: NEED, STRUCTURE, FEATURES, LIMITATIONS AND APPLICATIONS

- A **Mobile Database** is a database that can be connected to by a mobile computing device over a wireless mobile network.
- With mobile databases, users have access to corporate data on their laptop, PDA, or other Internet access device that is required for applications at remote sites.
- Examples: Sybase SQL Anywhere, Oracle Lite, Microsoft SQL Server Compact, SQLite, IBM DB2 Everyplace (DB2e).

Need:

- Need for data to be collected as it happens.
- Mobile data-driven applications enable us to access any data from anywhere, anytime.
- For example, Doctors can retrieve patient's medical history from anywhere; Salespersons can update sales records on the move; Reporters can update news database anytime.
- Mobile users must be able to work without a wireless connection due to poor or even non-existent connections.
- Need to access data while the mobile user is disconnected from the network—wireless or otherwise.
- This capability is best implemented by incorporating persistent data storage using a mobile database in your application components.

Components:

- The mobile database includes the following components:
  - The **main system database** that stores all the data and is linked to the mobile database.

  - The **mobile database** that allows users to view information even while on the move. It shares information with the main database.
  - **Mobile database platform** that includes a laptop, PDA, or other Internet access devices. This device uses the mobile database to access data.
  - A **two-way communication link** that allows the transfer of data between the mobile database and the main database.

Structure:

- The detailed structure of the mobile database is as follows:
- Mobile databases typically involve three parties: fixed hosts, mobile units, and base stations.
  1. **Fixed hosts** perform the transaction and data management functions with the help of database servers.
  2. **Mobile units** are portable computers that move around a geographical region that includes the cellular network (or "cells") that these units use to communicate to base stations.
  3. **Base stations** are two-way radios, installations in fixed locations that pass communications with the mobile units to and from the fixed hosts. They are typically low-power devices such as mobile phones, portable phones, or wireless routers.
- A cellular mobile network is similar to that of Mobile Network Architecture. It consists of Mobile Client (MC) containing data centric applications roaming between wireless cells and accesses a centralized database (fixed host). Some of the fixed hosts called Mobile Stations (MSSs) are augmented with wireless interfaces.

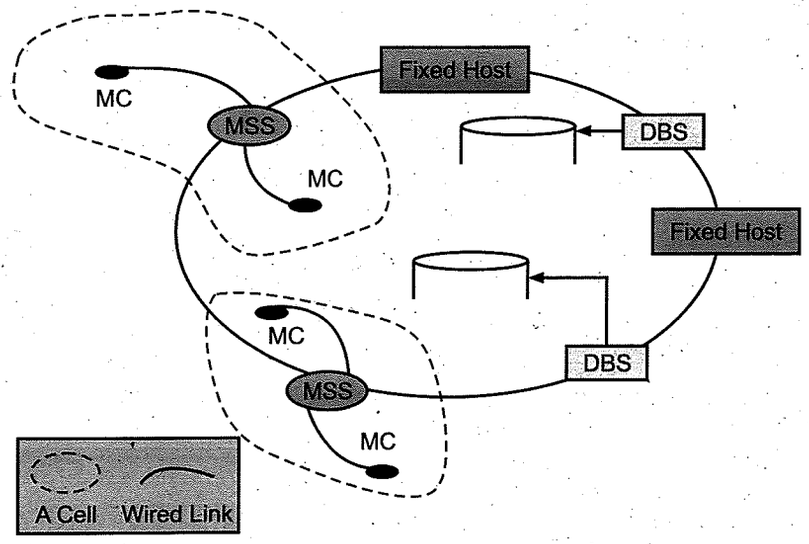


Fig. 6.13: Mobile Network Architecture

- The wireless channel is separated sub-channels: an uplink channel and a downlink channel. The uplink channel is used by submit queries, while the downlink channel is used by MSSs to answers from the server to target mobile client.

Advantages:

1. The data in a database can be accessed from anywhere using a mobile database. It provides wireless database access.
2. The database systems are synchronized using mobile databases and multiple users can access the data with seamless delivery process.
3. Mobile databases require very little support and maintenance.
4. The mobile database can be synchronized with multiple devices such as mobiles, computer devices, laptops etc.

Features:

- Mobile database provides relational database services but requires little memory.
- It analyzes and manipulates data on a mobile device.
- It handles local queries without connectivity.
- Users can choose to:
  - work online with a central DBMS server.
  - download data and work on them.
  - capture real-time data and Synchronize later.

Limitations:

- Some limitations of mobile databases are :
  - The mobile data is less secure than data that is stored in a conventional stationary database. This presents a security hazard.
  - The mobile unit that houses a mobile database may frequently lose power because of limited battery. This should not lead to loss of data in database.

Applications:

1. **Business:**
   - Information on customers, competitors, and market trends – anytime, anywhere.
   - Salespersons can update sales and customer data on the move.
2. **Public Sector:**
   - US Army uses mobile database technology to get current inventory and readiness info that can save them logistical costs.
3. **Health Sector:**
   - Used by Physicians to store and retrieve information while making their rounds.
   - Used by doctors and Paramedics to retrieve vital patient history & treatment information while attending to patients in battlefields and remote accident locations.

6.6 TEMPORAL DATABASES, TEMPORAL ASPECTS VALID TIME, TRANSACTION TIME OR DECISION TIME

- Databases that store information about states of the real world across time are called temporal databases.
- Temporal databases include all database applications that require some aspect of time when organizing their information.

Examples:

- There are many other examples of applications (reservation systems, scientific databases, company database, university database etc.) where some aspect of time is needed to maintain the information in a database.
  - 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 include the time when each data is measured.
  - A factory monitoring system may store information about current and past readings of sensors in the factory, for analysis.

6.6.1 Temporal Database Concepts

1. **Temporal Data Type:**

- The user-defined temporal data type is a time representation specially designed to meet the specific needs of the user.
- For example, the designers of a database used for class scheduling in a college might be based on a "Year: Term: Day: Period" format.
- Terms belonging to a user-defined temporal data type get the same query language support as do terms belonging to built-in temporal data types such as the DATE data type.

2. **Temporal aspects Valid Time:**

- Temporal databases allow the storage and the recovery of all the states assumed by an object during its lifetime, thus recording its evolution with time.
- When considering the issue of time in database systems, we must distinguish between time as measured by the system and time as observed in the real world.
- The valid time for a fact is the set of time intervals during which the fact is true in the real world.
- An example would be that Vijay Kumar was employed as marketing director at Matrix Incorporated from 1/3/1999 to 5/6/1999 i.e. the valid time interval for a tuple recording that Vijay Kumar was marketing director is from 1/3/1999 to 5/6/1999.

- The valid time has nothing to do with the time that data has been added to the database, so for example, we may have recorded this data and valid time about Vijay on 28/2/1999.
- The valid time for data can be changed e.g. perhaps Vijay Kumar comes out of retirement, and so a second interval from 12/10/2000 to 1/6/2001 is added to the valid times for his employment as marketing director.

3. Transaction time:

- The transaction time for a fact is the time interval during which the fact is current within the database system. This time is based on the transaction serialization order and is generated automatically by the system.
- Transaction-time values cannot be later than the current transaction time. Also as it is impossible to change the past, transaction times cannot be changed.

4. Timestamp:

- A timestamp is a time value associated with some object, e.g. an attribute value or a tuple. The concept may be specialised to valid timestamp, transaction timestamp or for a particular temporal data model, some other kind of timestamp.

5. Calendar:

- A calendar provides a human interpretation of time. As such, calendars assign meaning to temporal values where the particular meaning or interpretation is relevant to the user. In particular, calendars determine the mapping between human-meaningful time values and underlying time-line.
- Calendars are most often cyclic, allowing human-meaningful time values to be expressed briefly. For example, dates in the common Gregorian calendar may be expressed in the form <day, month, year> (for the UK) or <month, day, year> (for the US), where each of the fields 'day' and 'month' cycle as time passes (although year will continue to increase as time passes).

6. Time order:

- Different properties can be associated with a time axis composed from instants. Time is linear when the set of time points is completely ordered, also branching for the tasks of diagnosis, projection or forecasting (such as prediction of a medical evolution over time). Circular time describes recurrent events, such as "taking regular insulin every morning".

6.6.2 Types of Temporal Databases

- Temporal databases could be uni-temporal, bi-temporal or tri-temporal.
  1. A **uni-temporal database** has one axis of time, either the validity range or the system time range.
  2. A **bi-temporal database** has two axis of time:
     - valid time
     - transaction time or decision time

  3. A **tri-temporal database** has three axis of time:
     - valid time
     - transaction time
     - decision time

6.7 MULTIMEDIA DATABASE: ARCHITECTURE, TYPE AND CHARACTERISTICS

- A Multimedia database (MMDB) is a collection of related multimedia data.
- The multimedia databases are used to store multimedia data such as images, animation, audio, video along with text. This data is stored in the form of multiple file types like .txt(text), .jpg(images), .swf(videos), .mp3(audio) etc.
- Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. The variable length data structure cannot fit well into the relational framework, which mainly deals with fixed-format records.
- Furthermore, applications may require access to multimedia data on the basis of the structure of a graphical item or by following logical links. Conventional query languages were not designed for such applications.

Contents of the Multimedia Database:

- A Multimedia Database (MMDB) hosts one or more multimedia data types (i.e. text, images, graphic objects, audio, video, animation sequences). These data types are broadly categorized into three classes:
  1. Static media (time-independent: image and graphic object).
  2. Dynamic media (time-dependent: audio, video and animation).
  3. Dimensional media (3D game and computer aided drafting programs).
- Additionally, a Multimedia Database (MMDB) needs to manage additional information relating to the actual multimedia data. This is given in detail as follows :
  1. **Media data:** This is the multimedia data that is stored in the database such as images, videos, audios, animation, etc.
  2. **Media format data:** The Media format data contains the formatting information related to the media data such as sampling rate, frame rate, encoding scheme, etc.
  3. **Media keyword data:** This contains the keyword data related to the media in the database. For an image, the keyword data can be the date and time of the image, description of the image, etc.
  4. **Media feature data:** The Media feature data describes the features of the media data. For an image, feature data can be colours of the image, textures in the image, etc.
- The last three types are called *metadata* as they describe several different aspects of the media data. The media keyword data and media feature data are used as indices for searching purpose. The media format data is used to present the retrieved information.

6.7.1 Architecture

- Architecture of a multi-user database can become complex. It is not clear which architecture would be the best option for a multimedia database. A transaction involving multimedia data will in general be expected to take longer. Locks will have to be maintained for longer periods. MDBMS has formal database architecture. It has a separate user view from the system view.
- It basically constitutes three layer architecture. A simple representation is shown below:

Interface

- The interface between the user and the database is used for the following activities:
  - Object browsing
  - Compose
  - Dispose

Object Composition

- The object composition part of the multimedia database manages the multimedia objects.

Storage

- Storage functions of multimedia database involve clustering and indexing of multimedia data.

Three-layer architecture:

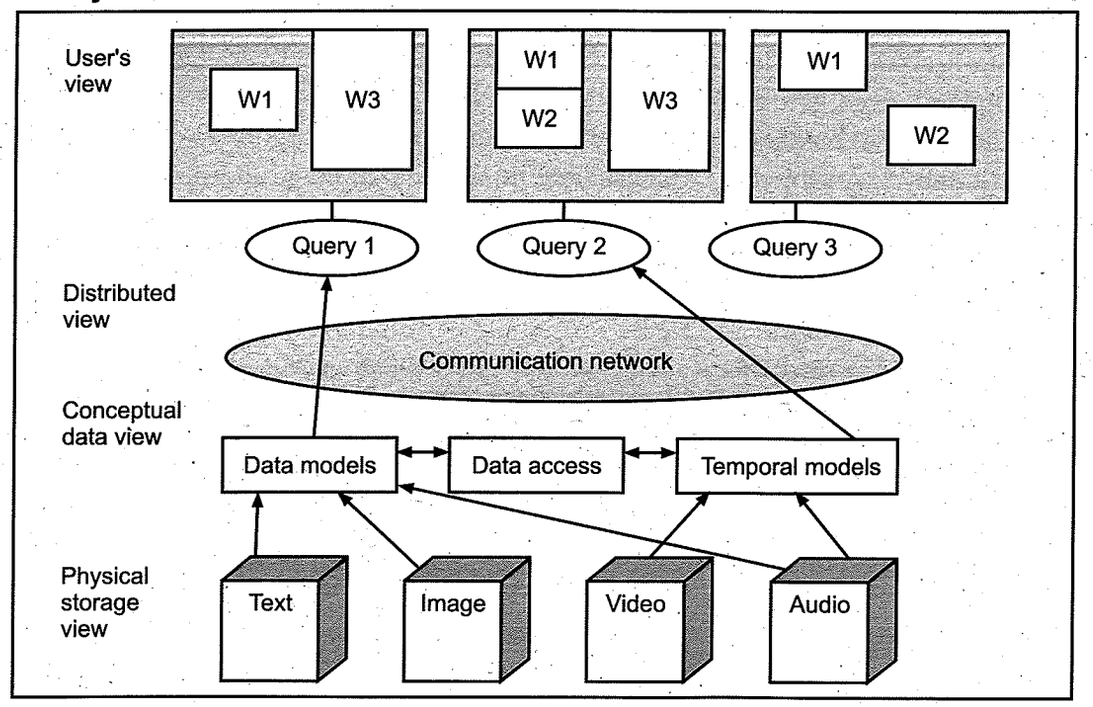


Fig. 6.14: Multimedia databases – User, Conceptual and Physical storage views

- A distributed MMDBMS has a more complex structure due to the database not being stored locally, at one place.

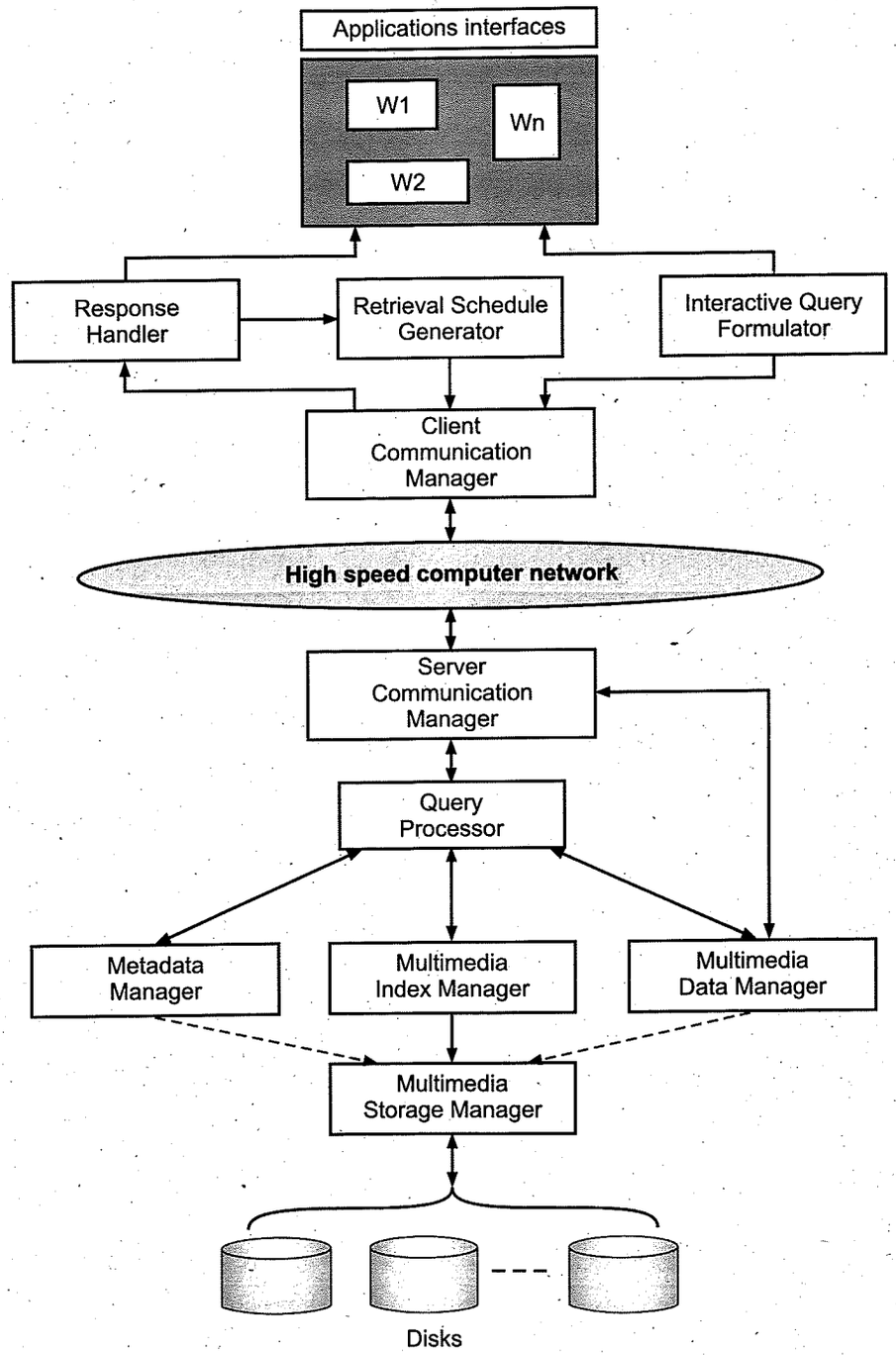


Fig 6.15: Distributed MMDBMS Architecture

- Fig. 6.15 shows the architecture of MMDBMS. This architecture comprises various components of the MMDBMS. The architecture shows multimedia database server and client connected by a computer network.
- There are six sub-components in MMDBMS server: 1. Storage Manager 2. Metadata Manager 3. Index Manager 4. Data Manager 5. Query Processor 6. Communication Manager.

1. **Storage Manager** handles the storage and retrieval of different media objects composing a database. The metadata as well as the index information related to media objects are also handled by this module.
2. **Metadata Manager** deals with creating and updating metadata associated with multimedia objects. It provides relevant information to the query processor in order to process a query.
3. **Index Manager** supports formulation and maintenance of faster access structures for multimedia information.
4. **Data Manager** helps in the creation and modification of multimedia objects and handling temporal and spatial characteristics of media objects.

   Metadata manager, index manager, and data/object manager access the required information through the storage manager.
5. **Query Processor** receives and processes user queries and utilizes information provided by metadata manager, index manager, and data manager for processing a query. When the user queries for the objects: the query processor identifies the sequence of the media objects to be presented and the sequence along with the temporal information for object presentation is passed back to the user.
6. **Communication Manager** manages the communication requirements of a multimedia database client. This module:
   - Depends on the services offered by the network service provider.
   - Reserves the necessary bandwidth for server-client communication.
   - Transfer queries and response between the server and the client.

- There are four sub-components in MMDBMS client: 1. Communication Manager 2. Retrieval Schedule Generator 3. Response Handler 4. Interactive Query Formulator.

1. **Communication Manager** manages the communication requirements of a multimedia database client. Its functionalities are the same as that of the server communication manager.
2. **Retrieval Schedule Generator** determines the schedule for retrieval media objects. For the retrieval schedule generator: the response handler provides information regarding the objects composing the response and the associated temporal information.

Retrieval Schedule Generator based on the available buffer and network throughput, determines the schedule for object retrieval. The retrieval schedule is used by the communication manager to download media objects in the specified sequence.

3. **Response Handler** interacts with the client communication manager in order to identify the type of response generated by the server.
   - If the response comprises of the information regarding the objects to be presented then the information is passed to the retrieval schedule generator for determining the retrieval sequence.
   - If the response comprises of the information regarding modifying the posed query or a null answer then the response is passed to the user.
4. **Interactive Query Formulator** helps a user to frame an appropriate query to be communicated to a database serve. It takes the response from the server through the response handler module in order to reformulate the query.

6.7.2 Types

- There are generally two types of multimedia databases: Linked Multimedia Databases and Embedded Multimedia Databases.

1. **Linked multimedia databases**

- Multimedia database can be organized as a database of metadata. This metadata links to the actual data such as graphic, image, animation, audio, sound etc. These data may store on Hard Disc, CD-ROM, DVD or Online.
- One great advantage of this type of database is that, the size of database will be small due to the reason that multimedia elements are not embedded in the database, but only linked to it.

2. **Embedded multimedia databases**

- Embedded Multimedia Database implies that the database itself contains the multimedia objects as in the binary form in the database. The main advantage of such kind of database is that retrieval of data will be faster because of the reduced data access time. However, the size of the database will be very large.

6.7.3 Application Areas

- The number of disciplines and enhance existing capabilities. Some of applications areas are:

1. **Documents and Records Management:** A large number of industries and businesses keep detailed records and a variety of documents. The data may include engineering design and manufacturing data, patient medical records, publishing material, and insurance claim records, to name a few.

2. **Knowledge Distribution:** The multimedia mode, a very effective means of knowledge distribution will encompass a phenomenal growth in electronic books, catalogs, manuals, encyclopaedias and repositories of information on many topics.
3. **Education and Training:** Computer-aided training and interactive learning materials for different audiences—from kindergarten students to equipment operators to professionals—can be designed from multimedia sources. Digital libraries are expected to have a major influence on the way future students and researchers as well as other users will access vast repositories of educational material.
4. **Marketing, Advertising, Retailing, Entertainment, and Travel:** There are virtually no limits to using multimedia information in these applications—from effective sales presentations to virtual tours of cities and art galleries. The film industry has already shown the power of special effects in creating animations and synthetically designed animals, aliens, and special effects. The use of predesigned stored objects in multimedia databases will expand the range of these applications.
5. **Real-time Control and Monitoring:** Coupled with active database technology, multimedia presentation of information can be very effective means for monitoring and controlling complex tasks such as manufacturing operations, nuclear power plants, patients in intensive care units and transportation systems.

6.7.4 Characteristics

1. **Corresponding Storage Media:** Data must be stored and managed according to their specific characteristics of the storage media.
2. **Descriptive Search Methods:** Query must be descriptive & content oriented.
3. **View Specific & Simultaneous Data Access:** Same data can be accessed through different queries by different applications.
4. **Management of Large Amounts of data:** The multimedia database requires a large size as the multimedia data is quite large and needs to be stored successfully in the database.
5. **Real time Data:** Data transfer of real time activity gets higher priority than other database activities.
6. **Large Transactions:** Large transactions must be done in a reliable fashion, since it takes long time.

Summary

- OODBMS is a DBMS where data is represented in the form of objects, as used in object-oriented programming.
- The OODBMS is based on three major components, namely: Object structure, Object classes, and Object identity.
- The Object ID (OID) is assigned by the system when the object is created, and cannot be changed.

- Spatial data represents the essential location characteristics of real or conceptual objects as those objects relate to the real or conceptual space in which they exist.
- A spatial index provides a mechanism to limit searches within tables (or data spaces) based on spatial criteria such as intersection, and containment.
- Mobile database is a database that can be connected to by a mobile computing device over a wireless mobile network.
- A temporal database is a collection of time-referenced data.
- Multimedia database can be organized as a database of metadata. This metadata links to the actual data such as graphic, image, animation, audio, sound etc.

Check Your Understanding

1. Which of the following is true concerning an ODBMS?
   (a) They have the ability to store complex data types on the Web.
   (b) They are overtaking RDBMS for all applications.
   (c) They are most useful for traditional, two-dimensional database table applications.
   (d) All of the above.
2. GIS deals with which kind of data _____.
   (a) Numeric data (b) Binary data
   (c) Spatial data (d) Complex data
3. What do mean by spatial data?
   (a) Interval-Data (b) Positional Data
   (c) Ordinal Data (d) Ratio Data
4. The form of data, having an associated time interval during which it is valid, is known as_____.
   (a) Temporal data (b) Snapshot data
   (c) Chunk data (d) Point in time data
5. Which of the following is the time of temporal data that record when a fact was recorded in a database?
   (a) Transaction time (b) Valid time
   (c) Enter time (d) Exit time
6. In mobile architecture, the base station covers a specific area that is called a _____.
   (a) Cell (b) Tessellate
   (c) Mobile station (d) None of the above
7. Which database management system is able to handle full text data, image data, audio and video?
   (a) Mobile (b) Graphics media
   (c) Multimedia (d) Relational

8. Which definition best describes an object?
   (a) Instance of a class (b) Instance of itself
   (c) Child of a class (d) Overview of a class
9. _____ represents an entity in the real world with its identity and behaviour.
   (a) A method (b) An object
   (c) A class (d) A Relation
10. Major components of The OODBMS are _____.
   (a) Object structure (b) Object classes
   (c) Object identity (d) All of the above
11. Which among the following is not a property of an object?
   (a) Identity (b) Properties
   (c) Attributes (d) Names

ANSWER KEY

1. (a)	2. (c)	3. (b)	4. (a)	5. (a)
6. (a)	7. (c)	8. (a)	9. (b)	10. (d)
11. (d)				

Practice Questions

Q.I : Answer the following questions in short.

1. What is OODBMS?
2. What is need of mobile database?
3. What is spatial data?
4. Define temporal databases.
5. What is OID?
6. What are the differences between valid time and transaction time?

Q.II: Answer the following questions.

1. Describe in detail OODBMS with its features.
2. Explain method of Spatial indexing with example.
3. Explain structure and need of mobile database.
4. What is multimedia database? Explain with its type.
5. Write in detail database design of OODBMS.

Q.III: Write a short note on:

1. Mobile database
2. Temporal aspects valid time
3. Transaction time or decision time
4. Objects
5. Attributes

❖❖❖

7... Crash Recovery and Backup

Objectives...

- To learn about Crash Recovery Process.
- To know about different types of failures in DBMS.
- To learn about Database backup and types of backups.

7.1 INTRODUCTION

- A computer system is subject to failure of various types. If some failure occurs during the execution of a transaction, it results in inconsistent database.
- Recovery system is an integral part of database system. It is responsible for restoration of the database to a consistent state that existed prior to the occurrence of the failure.

1. Failures and Errors:

- A system is reliable if it works as per its specifications and produces correct set of output values for a given set of input values.
- The **failure** of a system occurs when the system does not work according to its specifications and fails to deliver the service for which it was intended.
- An **error** in the system occurs when a component of a system assumes a state that is not desirable.
- A **fault** is detected either when an error is propagated from one component to another or the failure of the component is observed.

Note:

- Transaction failure doesn't result in loss of information.
- System crash loses the contents of volatile storage but it doesn't corrupt the non-volatile storage contents. (This is known as fail-stop assumption).

Note:

- Disk failure loses the data stored on disk. Copies of data on other disks or archival backups on tertiary media such as magnetic tapes are used to recover from the failure.

2. Recovery System:

- A computer system is electrical device, and causes different failures. There are various causes of failure including power failure, software error, disk crash, a fire in the machine room etc.
- In each of these failure cases, information may be lost. Therefore, the database system must take actions in advance to ensure that the durability and atomicity properties of transaction.
- An integral part of a database is a recovery scheme or system i.e. responsible for the restoration of the database to a consistent state that existed prior to the occurrence of the failure.

7.2 FAILURE CLASSIFICATIONS

- There are different types of failure that may occur in a system, each of failure needs to be deal with in a different manner.
- The simplest type of failure to deal with is one that does not result in the loss of information or data in the system.
- The failures that are more difficult or critical to deal with are those results in loss of information or data.

1. **Transaction failure:** There are two types of errors that may cause a transaction to fail. These are given below:
   (i) **System Error:** The system has entered an undesirable state as a result of which a transaction cannot continue with its normal execution. The transaction however can be re-executed at a later time.
   (ii) **Logical error:** The transaction can no longer continue with its normal execution; owing the some internal condition such as data not found, bad input or resource limit overflow exceeded.
2. **System crash:** There is a hardware bug or error 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 non-volatile storage remains intact, and is not corrupted.
3. **Disk failure:** In disk failures, disk block losses its content as a result of either a head failure during a data transfer operation. Copies of the data on other disks on tertiary media such as tapes, disks etc. are used to recover from the failure.

Methods to Control Failures:

- To determine how the system should recover from failure or crash, we need to identify the failure modes of those devices used for storing data. Then, we must consider how these failure modes or crash modes affect the contents or information of the database.
- Then we can use algorithms to ensure database consistency and transaction atomicity despite failures. These algorithms are known as Recovery Algorithms.
- Recovery algorithm have two parts:
  1. Actions taking during normal transaction processing to ensure enough information exists to recover from failures.
  2. Actions taking after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

7.3 STORAGE STRUCTURE

- To understand how to ensure the atomicity and durability properties of a transaction, we must gain a better understanding of these storage media and their access methods.

7.3.1 Storage Types

- There are different types of storage media according to their relative capacity, speed and flexibility to failure.
  1. **Stable storage:** Information in stable storage is never lost. The stable storage is theoretically impossible to obtain. It can be approximated by maintaining multiple copies on different non-volatile media. It survives all failures.
  2. **Volatile storage:** Information residing in this storage does not usually survive system crashes or failures. Access to this storage is extremely fast, both because of the speed of the memory access itself and because it is possible to access any data item in volatile storage directly. Examples: Main memory, Cache memory.
  3. **Non-volatile storage:** Information residing in this storage survives system crashes or failures. Examples: disks, magnetic tapes, flash memory. Tapes are used for archival storage whereas disks are used for on-line storage. Both, however, are subject to failure which may result in loss of information. Today non-volatile storage is slower than volatile storage by several orders of magnitude. In database systems, disks are used for most non-volatile storage. Other non-volatile media are normally used only for backup data. Flash storage though non-volatile has insufficient capacity for most database systems.
- Alternative forms of non-volatile storage, such as optical media, provide an even higher degree of reliability than do disks.

7.3.2 Stable Storage Implementation

- For implementation of stable storage, we need to replicate the required information/data on several non-volatile storage media with independent failure modes and to update the information in a controlled manner to ensure that failure during data transfer does not damage the needed information or data.
- Let us see how storage media can be protected from failure during data transfer.
- Block transfer between disk storage and memory can result in one of the following outcome:
  1. **Successful completion:** The transferred information/data arrived safely at its destination/output.
  2. **Partial failure:** A failure occurred in the middle of transfer process and the destination (output) block has incorrect information.
  3. **Total failure:** The failure occurred sufficiently early during the transfer process that the destination (output) block remains intact.
- If a data-transfer failure occurs, the system detects it and invokes a recovery procedure to restore the block to a consistent state. For this process, the system must maintain two physical blocks for each logical database block.
- In the case of mirrored disks, both blocks are at the same location. In the case of remote backup, one of the blocks is local, while the other block is at a remote site.

Protecting storage media from failure during data transfer:

- An output operation is executed as follows (assuming two copies of each block):
  1. Write the information onto the first physical block.
  2. When the first write completes successfully, write the same information on to the second physical block.
  3. The output is completed only after the second write completes successfully.
- During the recovery process, each pair of physical blocks is examined or noted. If both blocks are the same and no detectable error exists then no further actions are necessary.
- If one block contains a detectable error then we replace its content or data with the content or data of the second block. If both blocks contain no detectable error but they differ in content or data then we replace the content or data of the first block with the value of the second block. This type of recovery procedure ensures that a write to stable storage either succeeds completely or results in no change.
- The protocols for writing out a block to a remote site are similar to the protocols for writing blocks to a mirrored disk system.
- User can extend this procedure easily to allow the use of randomly large number of copies of each block of stable storage.

- Although a large number of copies reduce the probability of a failure to even lower than with two copies. It is usually reasonable to simulate stable storage with only two copies.

7.3.3 Data Access

- The database system resides permanently on non-volatile storage. They are partitioned into fixed-length storage units called *blocks*.
- Blocks are the units of data transfer to and from disk. It may contain several data items. We shall assume that no data item spans more than two blocks.
- Transactions input information/data 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*.
- Block movements between main memory and disk are initiated through the following two operations:
  1. **Input (B):** It transfers the physical block B to main memory.
  2. **Output (B):** It transfers the buffer block B to the disk and replaces the appropriate physical block there.

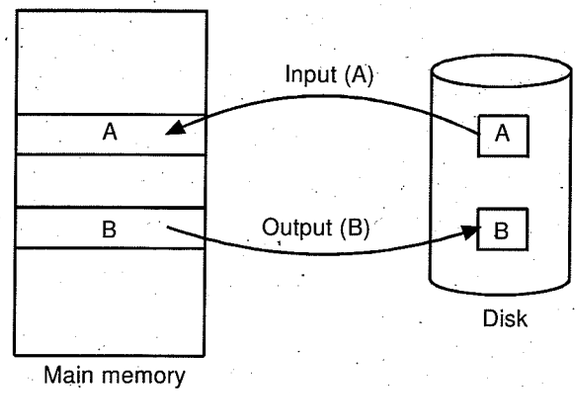


Fig. 7.1: Block storage operations

- Each T_i transaction has a private work area in which copies of all the data items updated and accessed by T_i are kept. This work area to T_i is created when the transaction is initiated; it is removed when the transaction either aborts or commits.
- Each data item x kept in the work area of T_i transaction is denoted by x_i. T_i transaction interacts with the database system by transferring data to and from its work area to the system buffer.

- We transfer data using two operations one is read (X) and other is write (X).
  1. **read (X):** It assigns the value of data item X to the local variable x_i. This read (X) is executed as follows:
     (a) If the block B_X on which X resides is not in main memory, then it issues input (B_X).
     (b) It assigns to x_i the value of X from the buffer block.
  2. **write (X):** It assigns the value of local variable x_i to data item X in the buffer block. Write (X) operation is executed as follows:
     (a) If block B_X on which X resides is not in main memory, then issue input(B_X).
     (b) Assign the value of x_i to X in buffer B_X.
- A buffer block is eventually written out to the disk either because the buffer manager needs the memory space for other processes or the database system wishes to reflect the change to B on the disk.
- When a transaction needs to access a data item X for the first time, it must execute read(X) operation. All updates to X are then performed on x_i. After the transaction accesses X for the final time, it must execute write(X) to reflect the change to X in the database itself.
- The output(B_X) operation for the B_X buffer block on which X resides does not need to take effect immediately after write(X) is executed, since the block B_X may contain other data items that are still being accessed.

7.4 RECOVERY & ATOMICITY

Recoverable Schedule:

- A recoverable schedule is defined as a schedule where for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.
- If some failure occurs during the execution of a transaction and database is in some inconsistent state following are two simple recovery procedures.
  (i) **Re-execute:** Execute the same transaction. But because of previous incomplete execution of that transaction, the database is already in an inconsistent state. Hence it results in an inconsistent database.
  (ii) **Do not re-execute:** If we do not re-execute the transaction, the database will remain in same inconsistent state. The atomicity property (perform either all or no database modifications) is not achieved. To achieve the goal of atomicity, information about modifications must be stored to stable storage before modifying the database. In this procedure, if some failures occur and modification information is complete then system can re-execute the transaction. Otherwise system will not re-execute the transaction.

Example:

- Consider the banking system and transaction T_i that transfers ₹ 50 from account A to account B, with initial values of A and B being ₹ 1000 and ₹ 2000, respectively. Suppose that a system crash has occurred during the execution of T_i, after output(BA) has taken place, but before output(BB) was executed, where BA and BB denote the buffer blocks on which A and B reside. Since the memory contents were lost, we do not know the fate of the transaction; thus, we could invoke one of two possible recovery procedures:
  1. **Re-execute T_i:** This procedure will result in the value of A becoming ₹ 900, rather than ₹ 950. Thus, the system enters an inconsistent state.
  2. **Do not re-execute T_i:** The current system state has values of ₹ 950 and ₹ 2000 for A and B, respectively. Thus, the system enters an inconsistent state.
- To achieve goal of atomicity, we must first output information describing the modifications of stable storage, without modifying the database itself. As we shall see, this procedure will allow us to output all the modifications made by a committed transaction, despite failures.
- Following are the two schemes to achieve the recovery from transaction failures:
  1. Log-based Recovery.
  2. Shadow Paging.

7.5 LOG BASED RECOVERY

- It assumes that transactions are executed serially i.e. only one transaction is active at a time. It uses a log to store the database modifications.

7.5.1 Log

- The most widely used structure for recording database modifications is the log. It resides in stable storage.
- Log is a sequence of log records and maintains a record of all the update activities in the database.
- There are several types of log records to record significant events during transaction processing.
  1. **Start of transaction:**
     denoted as; $<T_i$ start$>$
  2. **Update log:**
     It describes a single database write and it is denoted as:
     $<T_i, X_j, V_1, V_2>$
     where,
     $T_i \rightarrow$ Transaction identifier

$X_j \rightarrow$ Data item identifier

$V_1 \rightarrow$ Old value of X_j prior to the write operation

$V_2 \rightarrow$ New value of X_j after the write operation.

This means, Transaction T_i has performed a write on data item X_j . X_j had value V_1 before the write, and will have value V_2 after the write.

3. **Transaction commits:**
   denoted as: <T_i commit>
4. **Transaction abort:**
   denoted as: <T_i abort>

- There are two techniques for using log to achieve the recovery and ensure atomicity in case of failures.
  1. Deferred database Modification.
  2. Immediate database Modification.

7.5.2 Deferred Database Modification

- Deferred modification technique ensures transaction atomicity by recording all database modifications in the log, but deferring the execution of all write operations of transaction until the transaction partially commits.
- According to transaction state diagram, a transaction is in partially committed state when transaction completes executing the last instruction.
- The execution of transaction T_i proceeds as follows:
- Before T_i starts its execution record, a record <T_i start> is written to the log. A write(X) operation of T_i results the writing of a new record <T_i, X, V_2> to the log. When T_i partially commits, a record <T_i commit> is written to the log.

Note: It doesn't write V_1 - old value of X.

- When transaction T_i partially commits, the records associated with it in the log are used in executing the deferred writes. If system crashes before the transaction completes its execution or if the transaction aborts, then the information on the log is ignored. If some failure occurs while updating the database using log records, the log record is written in some stable storage. Hence, the transaction can resume its database modification. Using log, the system can handle any failure that results in loss of information on volatile storage. The recovery scheme uses the following recovery procedure.

  redo (T_i): It sets the value of all data items updated by transaction T_i to the new values. The set of data items and their respective new values can be found in the log.

  The redo operation must be idempotent i.e. executing it several times must be equivalent to executing it once.

A transaction can execute redo (T_i) if the log contains both the record <T_i start> and the record <T_i commit>.

Example: Consider in banking system, the transaction T_0 transfers ₹ 50 from account A to account B. Original values of account A and B are ₹ 1000 and ₹ 2000 respectively. This transaction is defined as:

```
T0   :   read (A);
         A:= A - 50;
         write (A);
         read (B);
         B:= B + 50;
         write (B)
```

- Consider the second transaction T_1 that withdraws ₹ 100 from amount C. This transaction is defined as:

```
T1   :   read (C)
         C:= C - 100
         write (C)
```

- These two transactions are executed serially <T_0, T_1>.
- The log containing relevant information on these two transactions is given below:

```
<T0 start>
<T0, A, 950>
<T0, B, 2050>
<T0 commit>
<T1 start>
<T1, C, 600>
<T1 commit>
```

- It shows the log that result from the complete execution of T_0 and T_1.
- The actual output can take place to database system in various orders. One such order is given below:

Log	Database
<T_0 start>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0 commit>	
	A = 900
	B = 2050

Log	Database
<T_1 start>	
<T_1, C, 600>	
<T_1 commit>	
	C = 600

- If system crashes before the completion of transactions:

Case 1: Crash occurs just after the log record for write (B) operation.

Log contents after the crash are:

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>

<T_0 commit> is not written; hence no redo operation is possible.

The values of amount remain unchanged i.e. amount A is ₹ 1000 and B is ₹ 2000.

Case 2: Crash occurs just after log record for write (C) operation.

The log contents after the crash are:

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 commit>
<T_1 start>
<T_1, C, 600>

- When the system comes back it finds <T_0 start> and <T_0 commit>. But there is no <T_1 commit> for <T_0 start>. Hence, system can execute redo (T_0) but not redo (T_1). Hence, the value of account C remains unchanged.

Case 3: Crash occurs just after the log record <T_1 commit>.

The log contents after the crash are:

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 commit>
<T_1 start>
<T, C, 600>
<T_1 commit>

- When system comes back it can execute both redo (T_0) and redo (T_1) operations.
- For each commit record, the redo (T_i) operation is performed. redo (T_i) writes the values to the database independent of the values currently in the database. Hence, the redo (T_i) is idempotent.

7.5.3 Immediate Database Modification

- The immediate update technique allows database modifications to be output to the database while the transaction is still in the active state. Data modifications written by active transactions are called uncommitted modifications. If a failure occurs during execution, the system must use the old value field of log records.
- Execution of transaction proceeds as follows:
  - Before T_i starts its execution, the record <T_i start> is written to the log.
  - Before executing any write(X) i.e. before modifying the database for write operation, it writes an update record <T_i, X, V_1, V_2> to the log.
  - When T_i partially commits, the record <T_i commit> is written to the log.
- Consider the same example of bank accounts and transactions T_0 and T_1. Transactions T_0 and T_1 are executed serially. The log corresponding to this execution is given below:

<T_0 start>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>
<T_0 commit>
<T_1 start>
<T_1, C, 700, 600>
<T_1 commit>

- The order in which output took place to both database system and log as a result of execution of T_0 and T_1 is:

Log	Database
<T_0 start>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
<T_0 commit> ⟶	A = 950 B = 2050
<T_1 start>	
<T_1, C, 700, 600> ⟶	C = 600
<T_1 commit >	

- Using the log, the system can handle any failure. The recovery scheme uses two recovery procedures:
  1. **Undo (T_i):** It restores the value of all data items updated by transaction T_i to the old values.
  2. **Redo (T_i):** It sets the value of all data items updated by transaction T_i to the new values.

- The undo (T_i) and redo (T_i) operations must be idempotent.
- Depending on the log, the recovery scheme determines which transaction need to be redone and which need to be undone.
- If the log record contains the record <T_i start> but does not contain the record <T_i commit> then T_i needs to be undone.
- If the log record contains both the records <T_i start> and <T_i commit> then T_i needs to be redone i.e. If a failure occurs before <T_i commit> the database modifications are rolled back by undo (T_i) operation. If a failure occurs after <T_i commit> then the transaction is re-executed by redo (T_i) operation.
- Consider the following conditions of failure for transactions T_0 and T_1:

Case 1: Failure occurs just after the log record for write(B) operation. Log contents and database are:

Log	Database
<T_0 start>	
<T_0 A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A = 950
	B = 2050

- Log contains <T_0 start> but does not contain <T_0, commit> record. Hence, transaction T_0 must be undone, so an undo (T_0) is executed. As a result, the values of accounts A and B are restored to 1000 and 2000 respectively.

Case 2: If some failure occurs just after log record for write(C), T_i has written to log. The log and database contents are:

Log	Database
<T_0 start>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A = 950
	B = 2050
<T_0 commit>	
<T_1 start>	
<T_1, C, 700, 600>	
	C = 600
<T_1 commit>	

- Log contains <T_0 start> and <T_0 commit>. Hence, the redo (T_0) is executed and values of accounts A and B are restored to the same (or new values) 950 and 2050 respectively. But the log doesn't contain <T_1 commit> for <T_1 start>. Hence, the value of account C to restored to old value by undo (T_1) operation. Hence value of C is ₹ 700.

Case 3: If system crashes just after the log record <T_1 commit> has been written to log.

The log and database contents are:

Log	Database
<T_0 start>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A = 950
	B = 2050
<T_0 commit >	
<T_1 start >	
<T_1, C, 700, 600 >	
	C = 600
<T_1 commit>	

- Log contains <T_0 start>, <T_0 commit> and <T_1 start> <T_1 commit> operations. Hence recovery system executes redo (T_0) and redo (T_1) operations. The values in accounts A, B and C are ₹ 950, ₹ 2050 and ₹ 600 respectively.

7.6 CHECKPOINT AND SHADOW PAGING IN DATA RECOVERY

7.6.1 Checkpoints

- When a system failure occurs, some transactions need to be redone and some need to be undone.
- Log record can determine this. But for that we need to search the entire log.
- There are two major difficulties with this approach:
  1. The search process is time consuming.
  2. Most of the transactions that will be redone have already written their updates into the database. Hence, it is better to avoid such redo operations.
- To reduce these types of overhead, checkpoints are introduced. During the execution the system maintains log using Immediate Database Modification Technique or Deferred Database Modification Technique.
- In addition, the system periodically performs checkpoints, which require following sequence of operations:
  1. Output onto stable storage all log records currently residing in main memory.

2. Output to the disk all modified buffer blocks.
3. Output onto stable storage a log record <checkpoint>.

- Transactions are not allowed to perform any update actions such as writing to a buffer block or writing a log record, while a checkpoint is in progress.
- The presence of <checkpoint> record in log allows the system to streamline its recovery procedure.
- After the failure has occurred, the recovery system examines the log to determine the most recent transaction T_i that started execution before the most recent checkpoint took place.
- It can find such a transaction by searching the log backward, from the end of the log, until it finds the first <checkpoint> record, then it continues the search backward until it finds the next <T_i start> record. This record identifies a transaction T_i.
- Once, the transaction is identified, redo or undo operations need to be applied to transaction T_i and all the transactions T_j that started executing after transaction T_i. The earlier part of transaction can be ignored. The recovery can be done by using immediate database modification or deferred database modification technique.

7.6.2 Shadow Paging

- Shadow paging is an alternative to log-based crash recovery technique. This is one possible form of indirect page allocation.
- **Paging:** Paging scheme is used in operating system for virtual memory management. The memory that is addressed by a process is called **virtual memory**.
- It is divided into pages that are assumed to be of a certain size (1 KB or 4 KB). The virtual or logical pages are mapped onto physical memory blocks of same size. The mapping of pages is provided by means of table called as **Page table**.

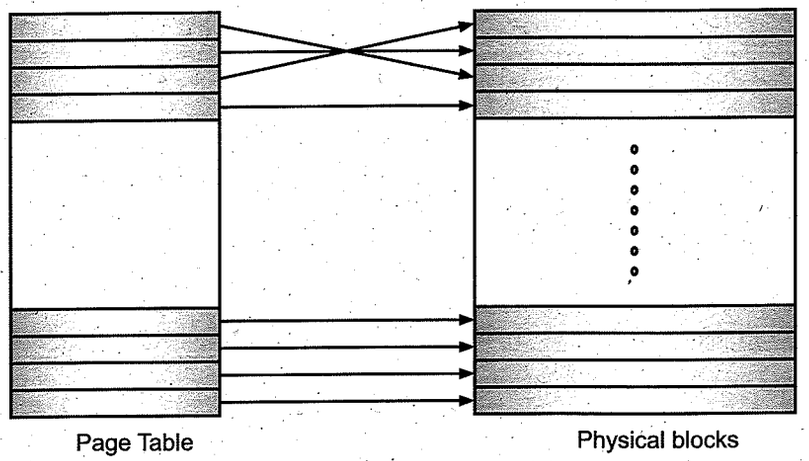


Fig. 7.2(a): Paging

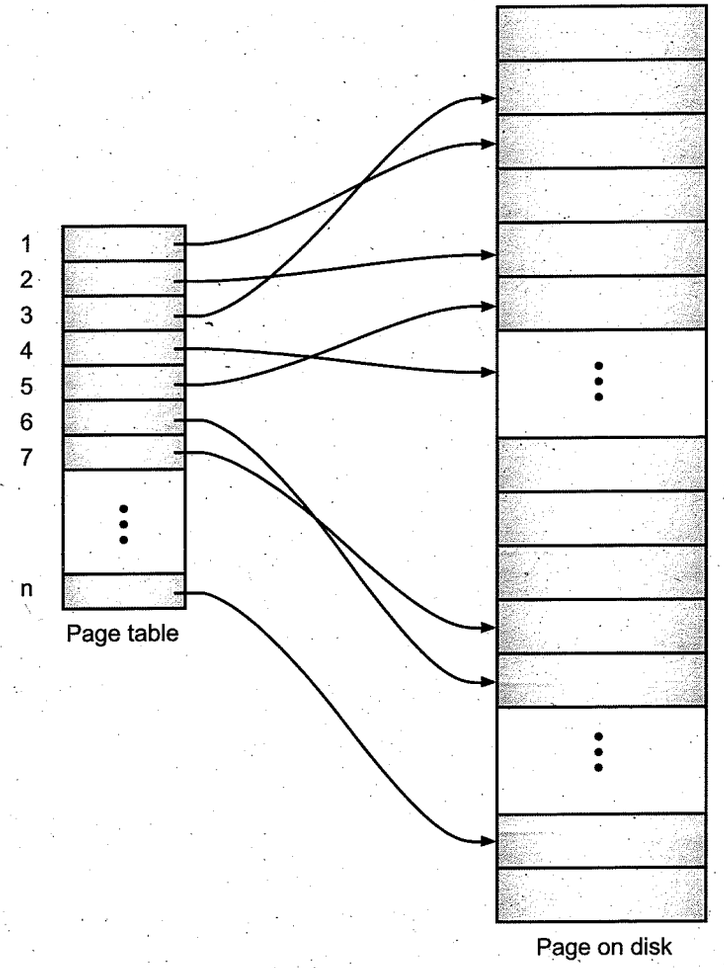


Fig. 7.2(b): Page Table

- Page table contains one entry for each logical page of the processes virtual address space. Page table is shown in the Fig. 7.2(b).
- The shadow page scheme uses following two page tables.
  1. Current page table.
  2. Shadow page table.

1. **Current page table :** Transaction addresses the database using current page table. It may change the current page table entries. The changes are made whenever the transaction executes write operation.
   - To modify a page, it copies that page to new blocks of physical storage. The page table entry corresponding to that page is made to point to new block of storage.
2. **Shadow page table :** The shadow page table is the original page table. It contains the entries that existed prior to the start of transaction. It remains unaltered by the transaction and it is used for undoing the transaction.

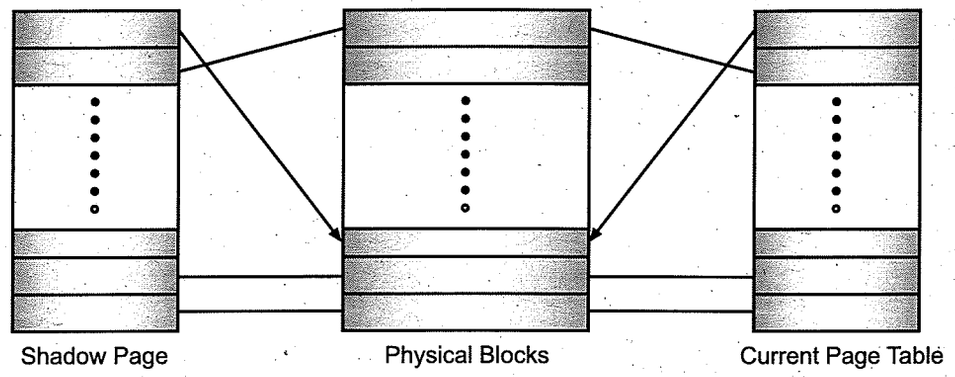


Fig. 7.3 (a): Before Starting the Transaction

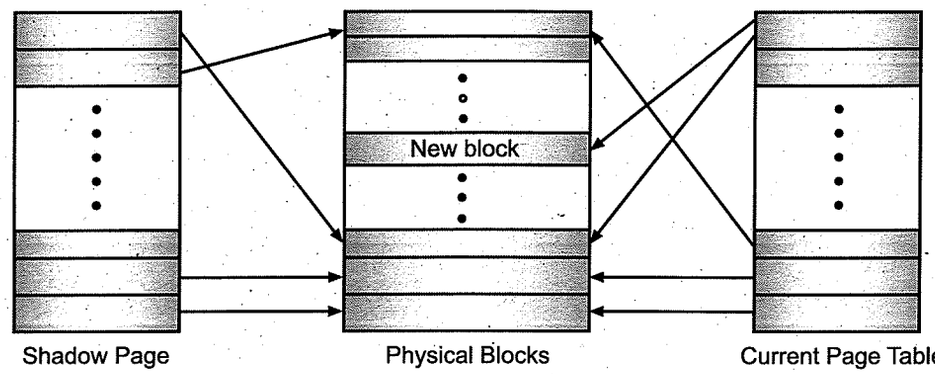


Fig. 7.3 (b): After Executing Write Operation

- Now, let us see how the transaction accesses data.
- The transaction uses the current page table to access the database blocks. The shadow paging scheme handles a write operation of transaction as follows:
  1. A free block of non-volatile storage is located from the pool of free blocks accessible by the database system.
  2. The block to be modified is copied onto this block.
  3. The original entry in the current page table is changed to point to this new block.
  4. The updates are propagated to the block pointed to by the current page table which in this case would be newly created block.
- Any changes made to the database are propagated to the blocks pointed to by the current page table. Once, a transaction commits, all modifications made by the transaction and still in buffers are propagated to physical database.
- It causes the current page table to be written to non-volatile storage. In case of system crash before the transaction commits, the shadow page table and the corresponding blocks containing the old database will continue to be accessible.

- The old database is made accessible just by modifying a single pointer from shadow page table to current page table.
- Once, the transaction completes its execution successfully, the shadow block can be returned to the pool of available non-volatile storage blocks to be used for further transactions.

Advantage of shadow paging scheme:

1. Recovery from system crash is relatively inexpensive and this is achieved without the overhead of logging.

Disadvantages of shadow paging scheme:

1. Over a period of time the database will be scattered over the physical memory and related records may require a very long access time.
2. When the transaction completed its execution, shadow blocks have to be returned to the pool of free blocks. If this is not done successfully, when a transaction commits, such blocks become inaccessible. This is called as **Garbage Collection Operation**.
3. The commit of a single transaction using shadow paging requires multiple blocks to be output the actual data blocks, the current page table, and disk address of the current page table log based schemes need to output only the log records.

7.7 DATABASE BACKUP AND TYPES OF BACKUPS

7.7.1 Database Backup

- A backup of a database is a way of protecting and restoring a file. It is accomplished by replication of the database and can be done for a database or database server.
- Database administrators can use the backup copy of the database to restore the database and its data and logs to its operating state. The backup of the database can be saved either locally or on a backup server.
- Also, all the business uses backups to protect their important database from disaster.
- Backup is a database utility. It creates backup copy of database by storing entire database on storage devices.

7.7.2 Types of Backup

- Backups can be divided into physical backups and logical backups.
  1. **Physical backups** are backups of the physical files used in storing and recovering your database, such as data files, control files, and archived redo logs. Ultimately, every physical backup is a copy of files storing database information to some other location, whether on disk or some offline storage such as tape.
  2. **Logical backups** contain logical data (for example, tables or stored procedures) exported from a database with an Oracle export utility and stored in a binary file,

for later re-importing into a database using the corresponding Oracle import utility.

Methods of Backup:

- The different methods of backup in a database are:

1. **Normal or Full Backup:** A full backup is the most basic of all backup types. A complete copy of all data is available in one location and restoration time is minimal. But this method takes a lot of time as the full copy of the database is made including the data and the transaction records.
2. **Incremental backup:** Incremental backup is a backup of all changes made since the last backup. With incremental backups, one full backup is done first and subsequent backup runs are just the changes made since the last backup. The result is a much faster backup then a full backup for each backup run. Storage space used is much less than a full backup and less then with differential backups. Restores are slower than with a full backup and a differential backup.
3. **Differential Backup:** This backup method is similar to full backup in that it stores both the data and the transaction records. However only that information is saved in the backup that has changed since the last full backup. Because of this, differential backup leads to smaller files.
4. **Remote Backup:** Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.

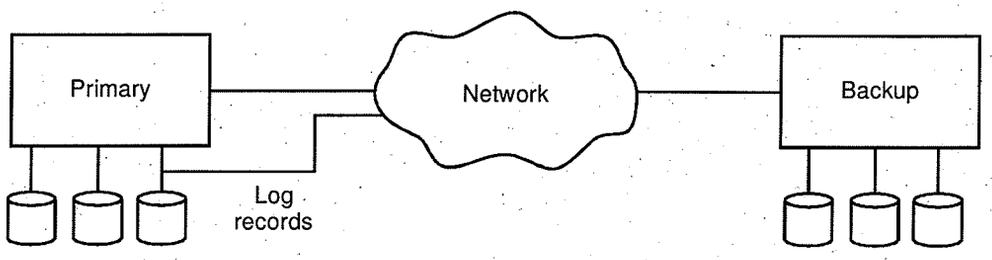


Fig. 7.4: Remote Backup System

- Several key points for designing remote backup system are listed below:

(i) **Time to recover:** For reducing delay in takeover, backup site periodically processes the redo log records performs a checkpoint. And it can then delete earlier parts of the log.

(ii) **Detection of failure:** The backup site must detect when primary site has failed to distinguish primary site failure from link failure maintain several communication links or paths between the primary and the remote backup.

(iii) **Alternative to remote backup:** In distributed database with replicated data remote backup is cheaper and faster, but less tolerant to failure.

(iv) **Transfer of control:** To take over control backup site first perform recovery using its copy of the database. When the backup site takes over processing it becomes the new primary. To transfer control back to old primary when it recovers, old primary must receive redo logs from the old backup.

5. **Cloud backup:** This term often used loosely and interchangeably with Online Backup and Remote Backup. This is a type of backup where data is backed up to a storage server or facility connected to the source via the Internet. With the proper login credentials, that backup can then be accessed securely from any other computer with an Internet connection. The term "cloud" refers to the backup storage facility being accessible from the Internet.

7.7.3 Methods to Store Data Backup

- You can store data in various ways:

1. **External Hard Drives**: Using an external hard drive for data backups for small businesses is the recommended method. As external hard drives are inexpensive as compared to tape drive systems. Easy to use you have to simply plug the hard drive into your computer's USB port.
2. **USB Drives:** The capacity of the USB drive is constantly increasing and is suitable for fast data backups. These have fast data transfer rates and are highly portable. You can conveniently backup or transfer offsite data to a USB drive. Since they do not have moving parts, they are very reliable for USB drives.
3. **Online Backup Services:** For more security always use strong passwords, and make sure to change them on a daily basis, and the backed up files are in encrypted form. But in case you lose your passwords, you can try the password recovery solutions present online.
4. **LAN Storage:** You can also back up files to another device or database if you have a local area network (LAN). If the system is located at the same location, however, it may be vulnerable to theft or damage. A server can be installed in a locked cage, drawer, to prevent theft.
5. **Tape Storage:** Tape storage help if you have large amounts of data to backup. They are highly reliable and can store massive quantities of data.
6. **Backup System:** Security of information is the best protection against such a disaster. Having a backup system that includes frequently and properly archiving and safeguarding your business data. There are many Data Backup tools available online.

Summary

- The failure of a system occurs when the system does not work according to its specifications and fails to deliver the service for which it was intended.
- An error in the system occurs when a component of a system assumes a state that is not desirable.
- A fault is detected either when an error is propagated from one component to another or the failure of the component is observed.
- The techniques used to recover the lost data due to system crash, transaction errors, viruses; catastrophic failure, incorrect commands execution etc. are database recovery techniques.
- Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- To recover from the loss of non-volatile memory, we restore the database from the archive and all the transactions that have been committed since the most recent dump are redone.
- A catastrophic failure is one where a stable, secondary storage device gets corrupt. With the storage device, all the valuable data that is stored inside is lost.
- A backup is a copy of data from your database that can be used to reconstruct that data. Backups can be divided into physical backups and logical backups.
- Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or real-time or online.

Check Your Understanding

1. An integral part of database that can restore database to consistent state of before failure is called _____.
   (a) Recovery scheme (b) Backup scheme
   (c) Restoring scheme (d) Transaction scheme
2. In log-based schemes, all updates are recorded on _____.
   (a) Blocks (b) Logs
   (c) Disks (d) Main memory
3. Occurrence of deadlock is an example of _____.
   (a) Application error (b) System crash
   (c) Logical error (d) System error
4. For recording database modifications, widely used structure is called _____.
   (a) Scheduling (b) Buffering
   (c) Log (d) Blocking

5. Errors like bad input, data not found, overflow, or resource limit exceeded are examples of _____.
   (a) View error (b) System crash
   (c) System error (d) Logical error
6. _____ is an alternative of log based recovery.
   (a) Disk recovery (b) Shadow paging
   (c) Dish shadowing (d) Crash recovery
7. The error that causes loss of the data of volatile storage, and halts the transaction processing is known as _____.
   (a) System error (b) Application error
   (c) System crash (d) Transaction error
8. Point out the correct statement.
   (a) Using differential backups can save available space
   (b) Using full backups can save available space
   (c) Using partial backups can save available space
   (d) None of the mentioned
9. Which of the following is not a recovery technique?
   (a) Deferred update (b) Immediate update
   (c) Two-phase commit (d) Recovery management
10. Checkpoints are a part of
    (a) Recovery measures (b) Security measures
    (c) Concurrency measures (d) Authorization measures

ANSWER KEY

1. (c)	2. (b)	3. (d)	4. (c)	5. (d)
6. (b)	7. (c)	8. (a)	9. (c)	10. (a)

Practice Questions

Q.I: Answers the following questions in short.

1. What is meant by crash recovery?
2. What is backup?
3. Describe storage structure.
4. Enlist various types of failures.
5. Explain the term Log-based recovery.

Q.II: Answers the following questions.

1. Explain Log-based recovery technique in details.
2. Explain deferred and immediate database modifications with example.
3. Enlist various storage types. Explain one of them in detail.
4. Describe failure classification in detail.
5. What is log? Explain in detail.

Q.III: Write a short note on:

1. Checkpoint
2. Buffer management.
3. Recovery and Atomicity
4. Failure with non-volatile storage.
5. Shadow Paging.

❖❖❖

8... Security and Privacy

Objectives...

- To learn about Database security issues.
- To study of Discretionary access control based on grant & revoking privilege.
- To know about Mandatory access control and Role-based access control for Multilevel security.
- To get information of Encryption & Public Key Infrastructure.

8.1 INTRODUCTION

- In this chapter, we will learn the techniques and methods used for protecting the database against users who are not authorized to access either certain parts of a database.
- **Database security** is a discipline that seeks methods to protect data stored at DBMSs from intrusions, improper modifications, theft, and unauthorized disclosure of private information.
- Data has to be protected against misuse, intentionally made inconsistency and also unintentionally made in consistency of data.

Techniques for database security:

- There are following techniques used for database security:
  1. **Discretionary security techniques** are used to grant privileges to uses including the capability to access specific data files, data records, data fields in a specified mode. (Read, Write).
  2. **Access control techniques** are used to make provisions for restricting access to the database system as a whole and handles user accounts and password to control the login process.

3. **Data encryption techniques** are used to protect sensitive data i.e. being transmitted via satellite and the data in the database is encoded using some coding algorithm.
4. **Mandatory security techniques** are used to enforce multilevel security by classifying the data and users into various security levels and then implementing the appropriate security policy of the organization.
5. **Statistical database security techniques** involve controlling the access to a statistical database which is used to provide statistical information of values based on various criteria.

8.2 DATABASE SECURITY ISSUES

- This section provides introduction to security issues the threats to databases and an overview of the countermeasures.
- Database security is a very large area that addresses many issues. Some of them are listed below:

1. **Policy issues:** At the institutional, governmental or corporate level as to what kinds of information should not be made publicly available: For example, personal employer records and credit ratings.
2. **System-related issues:** Such as the system levels at which different security functions should be enforced: For example, whether a security function should be handled at the operating system level or the DBMS level.
3. **Classified information issues:** The need in some organization to identity multiple security levels and to categories the users and data based on these classifications: For example, secret, top secret, confidential and unclassified etc. These classifications are only used on matters of national interest.
4. **Legal and Ethical issues:** They regarding the right to access certain information. Some information may be deemed to be private and cannot be accessed legally by unauthorized persons or users.

Threat:

- A **threat** may occur by a situation or event involving a person or the action or situations that are probably to bring harm to an organization and its database.
- The CIA triad : Confidentiality, Integrity, and Availability are the 3 key security goals of any information system.
- Threats to databases result in the degradation of some or all of the following security goals :

1. **Loss of availability:** Availability of database refers to making objects available to a human user to which they have a legitimate right. Administrators experience the loss of availability when a file server goes offline or a highly used database is out of service for one reason or another.

2. **Loss of confidentiality:** It is refers to the protection of data from unauthorized access. Examples of information that could be considered confidential are health records, financial account information, criminal records, source code, trade secrets, and military tactical plans.
3. **Loss of integrity:** Database integrity refers to the requirement that information/data be protected from improper modification. Modification of data includes insertion, creation, updation and deletion of information. Integrity is lost if unauthorized changes are made to the data by accidental acts. If the loss of data integrity is not corrected, continued used of the contaminated system or corrupted data could result in inaccuracy and fraud.

8.2.1 Security Countermeasures

- Security countermeasures are the controls used to protect the confidentiality, integrity, and availability of data and information systems.
- To protect databases against threats, following are the main countermeasures (control measures):

1. **Access control:** The security mechanism of DBMS must include some provisions for restricting access to the data base by unauthorized users. Access control is done by creating user accounts and to control login process by the DBMS. So, that database access of sensitive data is possible only to those people (database users) who are allowed to access such data and to restrict access to unauthorized persons.
   The database system must also keep the track of all operations performed by certain user throughout the entire login time.
2. **Inference control:** This method is known as the countermeasures to statistical database security problem. It is used to prevent the user from completing any inference channel. This method protects the sensitive information from indirect disclosure. Inferences are of two types, identity disclosure or attribute disclosure.
3. **Flow control:** This prevents information from flowing in a way that it reaches unauthorized users. Channels are the pathways for information to flow implicitly in ways that violate the privacy policy of a company are called covert channels.
4. **Encryption:** This method is mainly used to protect sensitive data (such as credit card numbers, OTP numbers) and other sensitive numbers. The data is encoded using some encoding algorithms.
   An unauthorized user who tries to access this encoded data will face difficulty in decoding it, but authorized users are given decoding keys to decode data.
5. **Authentication:** Authentication is the process of confirmation that whether the user log in only according to the rights provided to him to perform the activities of data base. A particular user can login only up to his privilege but he can't access the other sensitive data. The privilege of accessing sensitive data is restricted by using Authentication. By using these authentication tools for biometrics such as retina and figure prints can prevent the database from unauthorized/malicious users.

8.2.2 Role of Database Administrator (DBA)

- The Database Administrator (DBA) is the central authority for managing DBMS. Her/his responsibilities include granting privileges to users who need to use the system. DBA also responsible for classifying users and data in accordance with the policy of the organization.
- The Database Administrator has a DBA account in the DBMS, sometimes called a super user account or system account. This account provides powerful capabilities that are not made available to regular database users and accounts.
- DBA privileged commands (commands for granting and revoking privileges) to individual users accounts or user groups and for performing the following types of actions :
  1. **Privilege revocation:** Permits the DBA to revoke certain privileges that were previously given to certain accounts.
  2. **Account creation:** Creates a new account and password for a user to enable access to the Database.
  3. **Security level assignment:** Consists of assigning user accounts to the appropriate security classification level.
  4. **Privilege granting:** Permits the Database Administrator to grant certain privileges to certain accounts.
- Database Administrator is responsible for the overall security of the database system and overall control of the system. Action 2 in the abuse list is used to control access to the DBA as a whole, whereas actions 4 and 1 are used to control discretionary database authorization and action 3 is used to control mandatory authorization.

8.3 DISCRETIONARY ACCESS CONTROL BASED ON GRANT AND REVOKING PRIVILEGE

- Discretionary Access Control includes granting and revoking of the privileges which are known as Discretionary privileges.
- Let us consider privileges in the context of a Relational Database Management System. Many present relational DBMSs use some variation of this technique.
- The main idea is to include statements in the query language that allow the database Administrator and selected users to grant and revoke privileges.

8.3.1 Types of Discretionary Privileges

- There are two levels for assigning privileges to use the database system. They are :
  1. **Relation (or table) level:** In this level, DBA specifies particular privilege to access each individual relation or view in the database.
  2. **Account level:** In this level, DBA specifies particular privileges that each account holds independently of the relations in the database.

1. **Relation/Table Level:**
- This level of privilege applies to the relation level. This includes tables or relations and virtual relations known as views. Relational level privileges are defined for SQL2. The term relation may refer either to a base relation or to a view, unless we explicitly specify one or the other.
- Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns i.e. attributes of relations. SQL2 commands provide privileges at the relations and attribute level only.
- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the Access Matrix Model, where the rows of the matrix M represent subjects and the columns represent objects. Each position *m (i, j)* in the matrix represents the types of privileges that subject *i* holds on object *j*. This model specifies rights of each subject for each object.

2. **Account Level:**
- The privilege at the account level apply to the capabilities provided to the account itself. It can include:
  1. The CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation.
  2. The CREATE VIEW privilege.
  3. The ALTER privilege, to apply schema changes such as removing or adding attributes from relations.
  4. The DROP privilege, to delete relations or views.
  5. The MODIFY privilege, to update tuples, delete, or insert.
  6. The SELECT privilege, to retrieve information from the database by using a SELECT query.

8.3.2 Granting of Privileges

- Granting and revoking of privileges should be performed so that it ensures secure and authorized access and hence both of them should be controlled on each relation R in a database.
- It is carried out by assigning an owner account, which is the account that was used when the relation was created.
- The owner of a relation is the one who uses all privileges on that relation. In SQL2, the Database Administrator can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the CREATE SCHEMA command.
- The holder of owner account can pass privileges on any of the owned relations to other users by granting privileges to their accounts.
- In SQL, the following types of privileges can be granted on each individual relation R :

(i) **Select Privilege on R:** This privilege of SQL gives the account retrieval privilege. In SQL gives the account the privilege to use the select statement to retrieve tuples from relation R.

(ii) **Modify Privileges on R:** This privilege of SQL gives the account the capability to modify tuples of R. In SQL, this privilege is further divided into INSERT, DELETE, and UPDATE privileges to apply the corresponding SQL command to relation R.

(iii) **References Privilege on R:** This privilege of SQL gives the account the capability to reference relation R when specifying integrity constraints. This privilege can also be restricted to specific attributes of relation R.

8.3.3 Specifying Privileges using Views

- The mechanism of views is very important discretionary authorization mechanism in its own right.
- For example, if the owner X of a relation R wants another account Y to be able to retrieve only some fields of R, then X can create a view V or R that includes only those attributes and then grant SELECT on V to Y.
- The same applies to limiting Y to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that X wants to allow Y to access, (Refer example 8.1).

8.3.4 Revoking Privileges

- In some database cases, it is desirable to grant a privilege to a user temporarily.
- For example, the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed.
- Hence, a mechanism for revoking privileges is needed. In SQL a revoke command is used for the canceling privileges, (Refer Example 8.1)

8.3.5 Propagation of Privileges using the GRANT Option

- Whenever the owner X of a relation R grants a privilege on relation R to another account Y, the privilege can be given to Y with or without the GRNAT OPTION. If the GRANT OPTION is given, this means that Y can also grant that privilege on R to other accounts.
- Suppose that Y is given the GRANT OPTION by X and that Y then grants the privilege on relation R to a third account Z, also with GRANT OPTION. In this way, privileges on R can propagate to other accounts without the knowledge of the owner of relation R.
- If the owner account X now revokes the privilege granted to Y, all the privileges that Y propagated based on that privilege should automatically be revoked by the system.
- It is possible for a user to receive a certain privilege from more than two sources. For example, X4 may receive a certain UPDATE R privilege form both X2 and X3.
- In such case, if X2 revokes this privilege from X4; X4 will still continue to have the privilege by virtue of having been granted it from X3. If X3 later revokes the privilege form X4; X4 totally loses the privilege.

- Hence, a DBMS that allows propagation of privileges must keep track of how all the privileges were granted so that revoking of privileges can be done completely and correctly.

Example 8.1: Suppose that the DBA creates four accounts – X1, X2, X3 and X4. DBA wants only X1 to be able to create base relations; then the DBA must issue the following GRANT command in SQL:

```
GRANT CREATE TABLE TO X1;
```

The CREATE TABLE privilege gives account X1 the capability to create new database tables and is hence an account privilege.

Suppose that account X1 wants to grant to account X2 the privilege to insert and delete tuples in both of these relations. However, X1 does not want X2 to be able to propagate these privileges to additional accounts. X1 can issue the following command:

```
GRANT INSERT, DELETE ON STUDENT, DEPARTMENT TO X2;
```

STUDENT

NAME	S-id	B-DATE	ADDRESS	SEX	FEE – INSTALLMENT	D-NO

DEPARTMENT

D-NUMBER	D-NAME	MERS-id

Fig. 8.1: Schemas for two relations: Student and Department

- The owner account X1 of a relation automatically has the GRANT OPTION, allowing it to grant privileges on the relation to other accounts. However, account X2 cannot grant INSERT and DELETE privileges on the STUDENT and DEPARTMENT tables, because X2 was not given the GRANT OPTION in the preceding command.

```
GRANT SELECT ON STUDENT, DEPARTMENT TO X3 WITH GRANT OPTION;
```

- The clause WITH GRANT OPTION means that account X3 can now propagate the privilege to other accounts by using GRANT. For example, X3 can grant the SELECT privilege on the STUDENT relation to account X4 by issuing the following command :

```
GRANT SELECT ON STUDENT TO X4;
```

- X4 account cannot propagate the SELECT privilege to other accounts because the GRANT OPTION was not given to account X4.
- Suppose that account X1 decides to revoke the SELECT privilege on the STUDENT relation from X3; X1 then can issue this command:

```
REVOKE SELECT ON STUDENT FROM X3;
```

- The DBMS must now automatically revoke the SELECT privilege on STUDENT from account X4, because account X3 granted that privilege to account X4 and account X3 does not have the privilege any more.
- Suppose that account X1 wants to give back to account X3 a limited capability to SELECT from the STUDENT relation and wants to allow account X3 to be able to

propagate the privilege. The limitation is to retrieve only the NAME, B-DATE, and ADDRESS attributes and only for the tuples with D-NO = 5, account X1 then can create the following view :

```
CREATE VIEW X3 STUDENT AS
SELECT NAME, B-DATE, ADDRESS
FROM STUDENT
WHERE D-NO = 5;
```

- After the view is created, account X1 can grant SELECT on the view account X3 STUDENT to account X3 follows :

```
GRANT SELECT ON X3 STUDENT TO X3 WITH GRANT OPTION;
```

- Finally, suppose that account X1 wants to allow account X4 to update only the fee-installment attribute of STUDENT; account X1 can then issue the following command :

```
GRANT UPDATE ON STUDENT FEE-INSTALLMENT TO X4;
```

- The INSERT or UPDATE privilege can specify particular attributes that may be updated in a relation. Other privileges such as SELECT, DELETE are not attribute specific, because this specificity can easily be controlled by creating the appropriate views that include only the desired attributes and granting the corresponding privileges on the views.

8.3.6 Specifying Limits on Propagation of Privileges

- Limiting *horizontal propagation* to an integer number i means that an account Y given the GRANT OPTION can grant the privilege to at most i other accounts.
- *Vertical propagation* is more complicated. It limits the depth of the granting of privileges. Granting a privilege with a vertical propagation of zero is equivalent to granting the privilege with no GRANT OPTION.
- If account X grants a privilege to account Y with the vertical propagation set to an integer number $j > 0$, this means that the account Y has the GRANT OPTION on that privilege, but Y can grant the privilege to other accounts only with a vertical propagation less than j.
- In effect, vertical propagation limits the sequence of GRANT OPTIONS that can be given from one account to the next based on a single original grant of the privilege.

8.4 MANDATORY ACCESS CONTROL AND ROLE-BASED ACCESS CONTROL FOR MULTILEVEL SECURITY

8.4.1 Mandatory Access Control

- The discretionary access control method of granting and revoking privileges on relations has traditionally been the main security mechanism for Relational Database Systems (RDBMS).

- This technique is an all-or-nothing method: A user either has or does not have a certain privilege.
- In many database applications, an additional security policy is needed that classifies users and data based on security classes. This technique known as Mandatory Access Control.
- This technique would typically be combined with the discretionary access control mechanisms. It is very important to note that many commercial DBMSs currently provide mechanisms only for discretionary access control.

Security Levels:

- The need for multilevel security exists in military, Government and intelligence applications, as well as in many industrial and corporate applications.
- People and information are classified into different levels of trust and sensitivity. These levels represent the well-known security classifications.
- Typical security classifications are :
  1. Top secret (TS)
  2. Secret (S)
  3. Confidential (C)
  4. Unclassified (U)
- Where TS is the highest level and U the lowest. Other more complex security classification schemes are available, in which the security classes are organized in a lattice.
- For simplicity, we will use the system with four security classification levels, where $TS \geq S \geq C \geq U$, to illustrate our discussion.
- Classification of security levels for private organisations:
  - **Clearance level** indicates the level of trust given to a person with a security clearance, or a computer that processes classified information, or an area that has been physically secured for storing classified information. The level indicates the highest level of classified information to be stored or handled by the person, device, or location.
  - **Classification level** indicates the level of sensitivity associated with some information, like that in a document or a computer file. The level is supposed to indicate the degree of damage the country could suffer if the information is disclosed to an enemy.
  - **Security level** is a generic term for either a clearance level or a classification level.

The Bell-LaPadula Security Policy Model:

- The most commonly used model for multilevel security, is Bell-LaPadula model which classifies each subject (user, account, program) and object (relation, tuple, column, view, operation) into one of the security classification S, TS, C, U. We will refer to the clearance of a subject S as **class (S)** and to the classification of an object O as **class (O)**.

- Two rules are enforced on data access based on the subject/object classifications, these are given below :
  1. **Simple security property**: A subject S is not allowed read access to an object O unless

     class (S) ≥ class (O)
  2. **Star property** (or * property): A subject S is not allowed to write an object O unless

     class (S) ≤ class (O)

Example:

- Security labels in the military and government sectors consist of two components. A hierarchical component and a (possibly empty) set of categories.

A. The hierarchical component consists of the following, listed in decreasing order of sensitivity.
   1. Top Secret (TS)
   2. Secret (S)
   3. Confidential (C)
   4. Unclassified (U)

B. The set of categories consist of items such as NUCLEAR, CONVENTIONAL, NAVY, ARMY, NATO, etc.

- The label X is said to dominate label Y provided the hierarchical component of X is greater than or equal to the hierarchical component of Y, and the categories of X contain all the categories of Y.

For example:

- X = (TOP-SECRET, {NUCLEAR, ARMY}) dominates Y = (SECRET, {ARMY})
- X = (SECRET, {NUCLAR, ARMY}) dominates Y = (SECRET, {NUCLAR})
- X = (TOP-SECRET, {NUCLAR}) is incomparable to Y = (SECRET, {ARMY}),

i.e. neither one dominates the other.

- Note that two labels which dominate each other are exactly identical.
- Commercial organizations also use similar labels for protecting sensitive information. The main difference is that procedures for assigning clearances to users are much less formal than in the military or government sectors.

8.4.2 Difference between Discretionary Access Control and Mandatory Access Control

1. **Discretionary Access Control (DAC)** is characterized by a high degree of flexibility. DAC (Discretionary Access Control) is suitable for a large variety of application domains. The main disadvantage of DAC is their vulnerability to malicious attracts (Trojan horses embedded in application programs) for this reason discretionary authorization do not impose any control on how information is propagated and used once it has been accessed by user authorized to do so.

2. **Mandatory Access Control (MAC)** ensures a high degree of protection. Mandatory Access Control prevents any illegal flow of information. Mandatory Access Control suitable for military types of applications, which require a high degree of protection. However, they have the disadvantage of being too rigid in that they require a strict classification of objects and subjects into security levels and therefore, are applicable to very few environments. In many practical situations, Discretionary Access Control policies are preferred because they offer a better trade-off between applicability and security.

8.4.3 Role-Based Access Control

- **R**ole-**B**ased **A**ccess **C**ontrol also known as **RBAC** developed in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise wide systems.
- RBAC is the idea of assigning system access to users based on their role within an organization. The system needs of a given workforce are analyzed, with users grouped into roles based on common job responsibilities and system access needs. Access is then assigned to each person based strictly on their role assignment.
- For example, the purchase clerk might be defined as a role with access permissions for a subset of purchase ledger functionality and resources. As employees leave or join an organization then access control management is simplified to defining or revoking membership of the purchases clerk role.
- Roles can be created using the CREATE ROLE and delete or drop using DESTROY ROLE commands.
- The GRANT and REVOKE commands discussed under DAC then can be used to assign and revoke privileges form roles.
- In computer systems security, **R**ole-**B**ased **A**ccess **C**ontrol is an approach to restricting system access to authorized users. It is a newer alternative approach to Mandatory Access Control (MAC) and Discretionary Access Control (DAC). RBAC is sometimes referred to as Role Based Security.

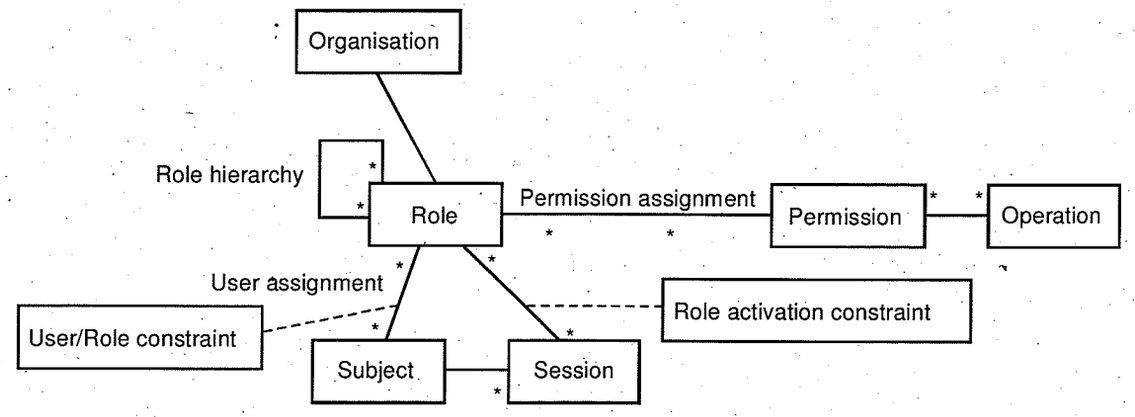


Fig. 8.2: Use of RBAC in IT organization

- Systems including Microsoft Active Directory, Microsoft SQL Server, SELinux, FreeBSD, Solaris, Oracle DBMS, PostgreSQL 8.1, SAP R/3 and many others effectively implement some form of RBAC.
- In an organization with a heterogeneous IT infrastructure and requirements that span dozens or hundreds of systems and applications, using RBAC to manage sufficient roles and assign adequate role memberships becomes extremely complex without hierarchical creation of roles and privilege assignments.
- Role hierarchy in Role-based access control is a natural way of organizing roles to reflect the organization's lines of responsibility and authority, by convention, junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy.
- The hierarchy diagrams are partial orders so they are reflexive, antisymmetric and transitive.

Advantages of RBAC :

1. It is solid. Companies can easily control users' access based on their roles.
2. It improves operational performance. In RBAC, many transactions are automated, and employees don't waste time using the applications and services that are not needed for fulfilling their responsibilities.
3. It decreases a risk of security breaches and data leakage because only a few people within an organization have access to sensitive data.
4. It has scalability. As a company grows and more employees are hired, the number of roles does not necessarily have to change. This makes it easier for the HR and IT departments, which otherwise would need to perform a number of administrative tasks.
5. It provides better security compliance. Having RBAC implemented means that a company meets requirements as far as privacy and confidentiality are concerned.

Implementation of RBAC:

RBAC can be achieved following these steps:

- Define data and resources to which access should be limited.
- Create roles with the same access needs.
- Avoid creating too many roles as in this case you will defeat the purposed and run a risk of creating a user-based access control instead of RBAC.
- Assign the roles with employees within your organization.
- Analyses how roles can be altered and how new employees can be registered and old accounts terminated.
- Ensure a company-wide RBAC that is integrated across all systems.
- Organize employee training so that the staff members are aware of the RBAC principles.
- Conduct audit to ensure that everything is followed through as planned.

8.5 ENCRYPTION AND PUBLIC KEY INFRASTRUCTURES

- Encryption is a method of maintaining secure data in an insecure environment.
- It consists of applying an encryption algorithm to data using some pre-specified encryption key.
- The resulting data/information has to be decrypted using a decryption key to recover the original data.
- In other words, "Encryption techniques/methods are used to provide security for highly sensitive data. In this method data will be stored in an encrypted form and this data can not be read, unless the user known to decrypt them".

8.5.1 Data Encryption Standard

- Data Encryption Standard (DES) has been widely accepted as a cryptographic standard. It can provide end-to-end encryption on the channel between the sender A and receiver B.
- The DES algorithm is a complex and careful combination of two of the fundamental building blocks of encryption :
  (i) Permutation
  (ii) Substitution.
- The DES algorithm derives its strength from repeated application of these two methods for a total of 16 cycles.
- Plaintext is encrypted as blocks of 64 bits. Although the key is 64 bits long, in effect the key can be any 56-bit number.
- After questioning the adequacy of DES, the NIST (National Institute of Standards) introduced the Advanced Encryption Standards (AES). AES algorithm has a block size of 128 bits, compared with DES's 56-block size.
- AES can use keys of 128 or 256 bits, compared with DES's 56-bit key. It introduces more possible keys, compared with DES and thus takes a much longer time to crack.

8.5.2 Digital Signature

- Digital Signature is an example of using encryption techniques to provide authentication services in e-commerce applications.
- A digital signature is a means of associating a mark unique to an individual with a body of text.
- It consists of a string of symbols. If a user's digital signature were always the same for each message, then one could easily fake if by simply copying the string of symbols.
- Thus, digital signatures must be different for each use, this can be achieved by making each digital signature a function of the message that it is signing, together with a time stamp.

- Public key techniques/methods are the best means of creating digital signatures with these properties.

8.5.3 Public Key Infrastructure

- A Public Key Infrastructure (PKI) is a combination of policies, procedures and technology needed to manage digital certificates in a public key cryptography scheme.
- PKI first emerged in the 1990s to help govern encryption keys through the issuance and management of digital certificates. These PKI certificates verify the owner of a private key and the authenticity of that relationship going forward to help maintain security. The certificates are akin to a driver's license or passport for the digital world.
- Common examples of PKI today are SSL certificates on websites so that site visitors know they're sending information to the intended recipient, digital signatures, and authentication for Internet of Things devices.

Building blocks of PKI:

1. **Symmetric encryption** algorithm is a simple cryptographic algorithm by today's standards. It involves the use of a single private cryptographic key to encrypt and decrypt information. It is one of the oldest methods of encryption, making it the most well-known. While using a single key makes the process faster, it lacks in security because it requires parties exchanging the key, making it more of a security risk.
2. **Asymmetric encryption** algorithm was developed to be more complex and secure than symmetric encryption. This process involves two keys, public and private, which are mathematically linked. One key encrypts and the other decrypts. The key owner will make one key open to the network (public) and keep the other key protected (private). If someone wants to send a message to the key owner, they can encrypt the message with the owner's public key, knowing that only the linked private key will be able to decrypt the message.

Components of a Public Key Infrastructure:

- The most important components of a public key infrastructure:

1. **Certification Authority (CA):** The CA is held by the key pair, including the secret key. Both keys have a mathematical relationship to one another, for example via a cryptographic hash function. The CA records each digital certificate using the secret key before verifying or issuing it to persons or companies. The persons or companies do not know the secret key, only the public key is known to them.
2. **Registration Authority (RA):** This is often called a subordinate CA. The RA is responsible for the registration of persons and companies. It allows the use of digital certificates for specific applications and also checks the certificates before they are issued by the CA.

3. **Directory and time stamp service:** All certificates and their public keys are stored here. Anyone can search this service for certificates to check if the certificates of certain people or companies are genuine, similar to a whitelist. In real-time, the certificates can also be checked for their validity in time in order to exclude expired certificates.
4. **Certificate Revocation List (CRL):** The CRL generates lists of invalid and rejected certificates whose keys are no longer secure. If the identity of the sender or recipient is not clear, certificates can be rejected. In such cases, the certificate is initially blocked and checked before it is permanently revoked. The CRL is a blacklist for certificates and associated digital signatures.
5. **X.509 certificates**: A digital certificate is an electronic data structure that binds an entity, being an institution, a person, a computer program, a web address etc., to its public key. Digital certificates are used for secure communication, using public key cryptography, and digital signatures. The purpose of a PKI is to make sure that the certificate can be trusted.

   The digital certificates in the PKI system are called X.509 certificates and are standardized. The authentication of a key is always bound to a sender or recipient, such as an email address or a domain name (DNS). Confidentiality is supposed to be guaranteed through the hierarchical structure of the certification. X.509 certificates contain various data concerning the cryptological hash function, which concerns the encryption of the public key and the validity of the digital signatures. In the latest version, X.509v3, extensions can also be implemented for specific applications.

Uses of PKI:

- A wide variety of use cases exist for PKI. Some of the most common PKI use cases include:
  1. SSL/TLS certificates to secure web browsing experiences and communications.
  2. Digital signatures on software.
  3. Restricted access to enterprise intranets and VPNs.
  4. Password-free Wi-Fi access based on device ownership.
  5. Email and data encryption.

8.5.3.1 Public Key Encryption

- Diffie and Hellman in 1976 proposed a new kind of cryptosystem, which they called Public Key Encryption.
- Public key algorithms are based on mathematical functions rather than operations on bit patterns. It involves the use of two separate keys (Private and Public) in contrast to conventional encryption, which uses only one key.

Components of Public Key Encryption :

- A public key encryption scheme has following components :
  - **(i) Plaintext:** Plaintext is the data or readable message that is fed into the algorithm as input.
  - **(ii) Encryption algorithm:** It performs various transformations on the plaintext.
  - **(iii) Public and Private keys:** These are a pair of keys that have been selected so that if one is used for encryptions, the other is used for decryption.
    - A Public Key is a cryptographic key that can be distributed to the public and does not require secure storage. Messages encrypted by the public key can only be decrypted by the corresponding private key.
    - Private Keys are used by the recipient to decrypt a message that is encrypted using a public key. Since the message is encrypted using a given public key, it can only be decrypted by the matching private key.
  - **(a) Cipher text:** Cipher text is the scrambled message produced as output. Cipher text depends on the plaintext and the key. For a given message, two different keys will produce two different cipher texts.
  - **(b) Decryption algorithm:** Decryption algorithm accepts the cipher text and the matching key and produces the original plaintext.
- The use of two keys can have profound consequences in the areas of confidentiality, key distribution and authentication

Public Key Cryptographic Algorithm:

- Public key cryptography is an application of asymmetric cryptography.
- A general purpose public key cryptographic algorithm relies on one key encryption and a different but related one for decryption.
- The essential steps are given below :
  - **Step 1:** If a sender (source) wishes to send a private message to a receiver (Destination), the sender encrypts the message using the receiver public key.
  - **Step 2:** When the receiver (Destination) receives the message, he/she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver known his / her private key.
  - **Step 3:** Each user generates a pair of keys to use for the encryption and decryption of messages.
  - **Step 4:** Each user places one of the two keys in a public register or other accessible file.

8.5.4 RSA Public Key Encryption Algorithm

- One of the first public key schemes RSA was introduced in 1978 by Adi Shamir, Len Adleman and Ron Rivest at MIT.
- RSA scheme is the most widely accepted and implemented approach to public key encryption.
- RSA scheme is incorporates results from number theory, combined with the difficulty of determining the prime factors of a target.
- It is also operates with modular arithmetic – mod n.
- Two keys, d and e are used by RSA for decryption and encryption. An important property is that they can be interchanged.

8.5.5 Statistical Database Security

- Statistical database are mainly used to produce statistics on various populations.
- The database may contain confidential data on individuals, which should be protected from user access.
- However, users are permitted to retrieve statistical information on the populations, such as sums, maximum, counts minimums averages and standard deviations.
- Statistical database security techniques must prohibit the retrieval of individual data this can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as AVERAGE SUM, MAX, MIN, COUNT and STANDARD DEVIATION. Such queries are sometimes called Statistical Queries.
- Statistical queries involve applying statistical functions to a population of tuples. For example, user may want to retrieve the number of individuals in a population or the average income in the population.
- Statistical users are not allowed to retrieve individual data, such as the income of specific person.

Summary

- Database security is a discipline that seeks methods to protect data stored at DBMSs from intrusions, improper modifications, theft, and unauthorized disclosure of private information.
- Database security can include the secure management of encryption keys, protection of the encryption system, management of a secure, off-site encryption backup, and access restriction protocols.
- The usual way of supplying access controls to a database system is dependent on the granting and revoking of privileges within the database. A privilege allows a user to create or access some database object or to run some specific DBMS utilities.

- Threats to databases can result in the loss or degradation of some or all of the following commonly accepted security goals: Confidentiality, Integrity, and Availability.
- Database confidentiality refers to the protection of data from unauthorized disclosure. Database integrity refers to the requirement that information be protected from improper modification.
- Database integrity refers to the requirement that information be protected from improper modification.
- The database provides various types of access controls: Discretionary Access Control (DAC) and Mandatory Access Control (MAC).
- Discretionary security mechanisms are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- Mandatory security mechanisms are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization.
- Public Key Infrastructure (PKI) is a technology for authenticating users and devices in the digital world.
- Encryption is a process that encodes a message or file so that it can be only be read by certain people.
- Data, or plaintext, is encrypted with an encryption algorithm and an encryption key. The process results in cipher text.

Check Your Understanding

1. Data security threats include :
   (a) privacy invasion (b) hardware protection
   (c) fraudulent manipulation of data (d) all of the above
2. Data integrity means :
   (a) providing first access to stored data.
   (b) ensuring correctness and consistency of data.
   (c) providing data sharing.
   (d) none of the above
3. Authentication refers to :
   (a) methods of restricting user access to system.
   (b) controlling access to portions of database.
   (c) controlling the operation on the data.
   (d) all of the above

4. What are the elements of the CIA Triad?
   (a) Confidentiality, integrity, and availability
   (b) Confidentiality, interest, and accessibility
   (c) Control, integrity, and authentication
   (d) Calculations, interpretation, and accountability
5. _____ access controls rely upon the use of labels.
   (a) Discretionary (b) Role-based
   (c) Mandatory (d) Nondiscretionary
6. Rule-Based Access Control (RuBAC) access is determined by rules. Such rules would fit within which category of access control?
   (a) Discretionary Access Control (DAC)
   (b) Mandatory Access control (MAC)
   (c) Non-Discretionary Access Control (NDAC)
   (d) Lattice-based Access control
7. What is called the type of access control where there are pairs of elements that have the least upper bound of values and greatest lower bound of values?
   (a) Mandatory model (b) Discretionary model
   (c) Rule model (d) Lattice model
8. Which of the following public key distribution systems is most secure?
   (a) Public-Key Certificates
   (b) Public announcements
   (c) Publicly available directories
   (d) Public-Key authority
9. In cryptography, what is cipher?
   (a) algorithm for performing encryption and decryption.
   (b) encrypted message
   (c) both (a) and(b)
   (d) decrypted message
10. What is data encryption standard (DES)?
    (a) block cipher (b) stream cipher
    (c) bit cipher (d) byte cipher

ANSWER KEY

1. (d)	2. (b)	3. (d)	4. (c)	5. (c)
6. (c)	7. (d)	8. (a)	9. (a)	10. (a)

Practice Questions

Q.I: Answers the following questions in short.

1. What is database security?
2. Which are commonly accepted database security goals?
3. Define PKI in encryption.
4. What is encryption?
5. Which are methods of access control?
6. What is meant by granting a privilege?

Q.II: Answers the following questions.

1. What is the difference between discretionary and mandatory access control?
2. Describe various levels of security in detail.
3. What are the typical security classifications?
4. What is meant by revoking a privilege? Describe in detail.
5. Explain the theory of discretionary access control in a database system.

Q.III: Write a short note on:

1. Digital signature
2. Database security issues
3. RSA algorithm
4. Data and Advanced Encryption Standards
5. Role of DBA in database security

❖❖❖

9... No-SQL Database

Objectives...

- To learn about NoSQL Database and its types.
- To know about Need of NoSQL Database.

9.1 INTRODUCTION

- The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.
- **NoSQL**, which stand for "not only SQL," is an alternative to traditional relational **databases** in which data is placed in tables and data schema is carefully designed before the **database** is built. The NoSQL movement has been in the news in the past few years as many of the Web 2.0 leaders have adopted a NoSQL technology. Companies like Facebook, Twitter, Digg, Amazon, LinkedIn and Google all use NoSQL in one way or another.

9.1.1 What is NoSQL?

- NoSQL is a non-relational database management system, different from traditional relational database management systems in some significant ways.
- **NoSQL databases** are especially useful for working with large sets of distributed data. (For example, Google or Facebook which collects terabits of data every day for their users).
- These types of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

9.1.2 Characteristics of NoSQL

- Characteristics of NoSQL database are give below:

1. **Schema-free:** A database schema is the structure of database system, described in a formal language supported by the database management system. In a relation database the schema defines the tables, the fields in each table and the relationships

between fields and tables, generally it stored in a data dictionary. In NoSQL, collection is a group of documents where document represent a row and collection represent a table in a relational database. Collections are schema free which means within a single collection different types and structured of documents can be stored. For example, both of the following documents can be stored in a single collection.

{"colour" : "red"}

{"salary" : 100.00}

Note that the previous documents have no common data elements at all. This flexibility means that schema free practice can be possible in NoSQL database.

2. **Horizontally scalable:** To scale horizontally (or scale out) means to add more nodes to a system (such as adding a new computer). In NoSQL system, data store can be much faster as it takes advantage of "scaling out" which means to add more nodes to a system and distribute the load over those nodes.

9.2 TYPES OF NoSQL

Types of NoSQL Database:

1. **Key Value Store:**
   - These are the simplest NoSQL databases. Every single item in the database is stored as an attribute name or key together with its value.
   - Example: Memcached, Redis(supports multiple data structures).
   - The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data. A bucket is a logical group of keys but they don't physically group the data. There can be identical keys in different buckets.
   - To read a value you need to know both the key and the bucket because the real key is a hash (Bucket+ Key).
   - There is no complexity around the Key Value Store database model as it can be implemented in a breeze. Not an ideal method if you are only looking to just update part of a value or query the database.

Table 9.1: Keys and Values

Key	Value
"India"	{"A-35, Sector-57, Pune , India – 4110909"}
"US"	{"3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033"}

2. **Document Store:**
   - It pairs each key with a complex data structure known as document. It can contain many different key value pairs or key array pairs or even nested documents.

   - Example: MongoDB
   - The data which is a collection of key value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as "documents") provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects) are some common standard encodings.
3. **Graph Store:**
   - These are used to store information about networks, such as social connections.
   - Example: Neo4j. This database built for data sets that contain strong relationships and connections. Widely used in the industry in companies such as eBay and Walmart.
   - In a Graph Base NoSQL Database, a flexible graphical representation is used which is perfect to address scalability concerns. Graph structures are used with edges, nodes and properties which provides index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database.

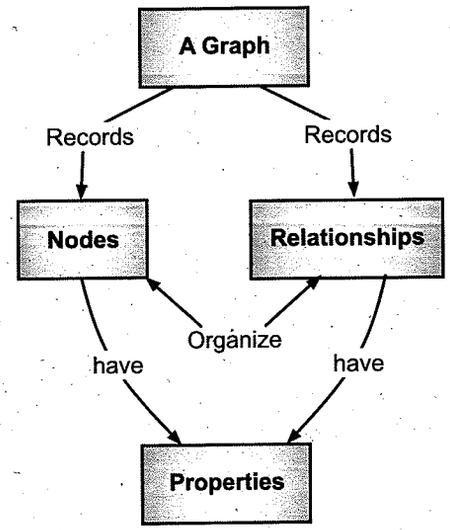


Fig. 9.1: Graph Store NoSQL

   - Databases that uses edges and nodes to represent and store data.
   - Nodes are organized by some relationships with one another, which are represented by edges between the nodes.
   - Both the nodes and the relationships have some defined properties.
4. **Column Store:**
   - In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns

that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows.

- Example: The most popular column-based NoSQL DBMS is Cassandra.
- In comparison, most relational DBMS store data in rows, the benefit of storing data in columns, is fast search/access and data aggregation. Relational databases store a single row as a continuous disk entry. Different rows are stored in different places on disk while Columnar databases store all the cells corresponding to a column as a continuous disk entry thus makes the search/access faster.

9.3 NEED OF NoSQL DATABASES

- NoSQL Database used for following reasons:

1. NoSQL offers a simpler data model. In other words, it motivates to use the concepts of embedding and indexing your data rather than using the concept of joining the data.
2. NoSQL will be useful when a developer wants to do rapid development of the application.
3. It provides high scaling out capability.
4. It is flexible so it allows you to add any kind of data in your database.
5. It also provides distributed storage and high availability of the data.
6. Streaming is also accepted by NoSQL because it can handle a high volume of data which is stored in your database.
7. It offers real-time analysis, and redundancy so as to replicate your data on more than one server.
8. It is highly scalable therefore it can be implemented with a very low budget.

9.4 ADVANTAGES AND DISADVANTAGES OF NoSQL DATABASE

9.4.1 Advantages

- Advantages of NoSQL database are given below:

1. The NoSQL approach presents huge advantages over SQL databases because it allows one to scale an application to new levels. The new data services are based on truly scalable structures and architectures built for the cloud, built for distribution, and are very attractive to the application developer.
2. There's no need for DBA, no need for complicated SQL queries and it is fast.
3. This is no small matter — a good programmer's freedom to choose a data model, write a program or an application with familiar tools, reduce dependencies on other people, test and optimize the code without doing guesswork or counting on a black box (DB).

9.4.2 Disadvantages

- There are some disadvantages to the NoSQL approach, these are:

1. **At the system level, data models are key**: Not having a skilled authority to design a single, well-defined data model, regardless of the technology used, has its drawbacks. The data model may suffer from duplication of data objects (non-normalized model). This can happen due to the different object model used by different developers and their mapping to the persistency model. At the system level one must also understand the limitations of the chosen data service, whether it is size, operations per second, concurrency model, etc.
2. **At the architecture level, two major issues are interfaces and interoperability:** Interfaces for the NoSQL data services are yet to be standardized. Even DHT, which is one of the simpler interfaces, still has no standard semantics, which includes transactions, none blocking API etc. Each DHT service used comes with its own set of interfaces. Another big issue is how different data structures, such as DHT and a binary tree, just as an example, share data objects. There are no intrinsic semantics for pointers in all those services. In fact, there's usually not even strong typing in these services. It's the developer's responsibility to deal with that.
3. **Interoperability** is an important point, especially when data needs to be accessed by multiple services.

9.5 DIFFERENCE BETWEEN RDBMS AND NoSQL DATABASE

- Following is a list of the differences between NoSQL and RDBMS:

1. In terms of data format, NoSQL does not follow any order for its data format. Whereas, RDBMS is more organized and structured when it comes to the format of its data.
2. When it comes to scalability, NoSQL is more very good and more scalable. Whereas, RDBMS is average and less scalable than NoSQL.
3. For querying of data, NoSQL is limited in terms of querying because there is no join clause present in NoSQL. Whereas, querying can be used in RDBMS as it uses the structured query language.
4. The difference in the storage mechanism of NoSQL and RDBMS, NoSQL uses key-value pairs, documents, column storage, etc. for storage. Whereas, RDBMS uses various tables for storing data and relationships.

9.6 USE CASES

- For organizations facing NoSQL versus relational decisions of their own, here are some use cases where a NoSQL database may have a legitimate role to play:
- **Personalization:** A personalized experience requires data, and lots of it – demographic, contextual, behavioral and more. The more data available, the more personalized the experience. However, relational databases are overwhelmed by the volume of data required for personalization. In contrast, a distributed NoSQL database can scale elastically to meet the most demanding workloads and build and update visitor profiles on the fly, delivering the low latency required for real-time engagement with your customers.
- **Profile Management:** User profile management is core to Web and mobile applications to enable online transactions, user preferences, user authentication and more. Today, Web and mobile applications support millions or even hundreds of millions of users. While relational databases can struggle to serve this amount of user profile data as they are limited to a single server, distributed databases can scale out across multiple servers. With NoSQL, capacity is increased simply by adding commodity servers, making it far easier and less expensive to scale.
- **Real-Time Big Data:** The ability to extract information from operational data in real-time is critical for an agile enterprise. It increases operational efficiency, reduces costs, and increases revenue by enabling you to act immediately on current data. In the past, operational databases and analytical databases were maintained as different environments. The operational database powered applications while the analytical database was part of the business intelligence and reporting environment. Today, NoSQL is used as both the front-end – to store and manage operational data from any source, and to feed data to Hadoop – as well as the back-end to receive, store and serve analytic results from Hadoop.
- **Content Management:** The key to effective content is the ability to select a variety of content, aggregate it and present it to the customer at the moment of interaction. NoSQL document databases, with their flexible data model, are perfect for storing any type of content – structured, semi-structured or unstructured – because NoSQL document databases don't require the data model to be defined first. Not only does it enable enterprises to easily create and produce new types of content, it also enables them to incorporate user-generated content, such as comments, images, or videos posted on social media, with the same ease and agility.
- **Catalog:** Catalogs are not only referenced by Web and mobile applications, they also enable point-of-sale terminals, self-service kiosks and more. As enterprises offer

more products and services, and collect more reference data, catalogs become fragmented by application and business unit or brand. Because relational databases rely on fixed data models, it's not uncommon for multiple applications to access multiple databases, which introduces complexity and data management challenges. By contrast, a NoSQL document database, with its flexible data model, enables enterprises to more easily aggregate catalog data within a single database.

- **Customer 360° View:** Customers expect a consistent experience regardless of channel, while the enterprise wants to capitalize on upsell/cross-sell opportunities and to provide the highest level of customer service. However, as the number of products and services, channels, brands and business units increases, the fixed data model of relational databases forces enterprises to fragment customer data because different applications work with different customer data. NoSQL document databases use a flexible data model that enables multiple applications to access the same customer data as well as add new attributes without affecting other applications.
- **Mobile Applications**: With nearly two billion smartphone users, mobile applications face scalability challenges in terms of growth and volume. For instance, it is not uncommon for mobile games to reach tens of millions of users in a matter of months. With a distributed, scale-out database, mobile applications can start with a small deployment and expand as the user base grows, rather than deploying an expensive, large relational database server from the beginning.
- **Internet of Things:** Today, some 20 billion devices are connected to the Internet – everything from smartphones and tablets to home appliances and systems installed in cars, hospitals and warehouses. The volume, velocity, and variety of machine-generated data are increasing with the proliferation of digital telemetry, which is semi-structured and continuous. Relational databases struggle with the three well-known challenges from big data IoT applications: scalability, throughput, and data variety. By contrast, NoSQL allows enterprises to scale concurrent data access to millions of connected devices and systems, store large volumes of data, and meet the performance requirements of mission-critical infrastructure and operations.
- **Digital Communications:** In an enterprise environment, digital communication may take the form of online interaction via direct messaging to help visitors find a product or complete the checkout process. And as with mobile text messaging, the application may need to support millions of website visitors. Relational databases are limited in responsiveness and scalability while NoSQL databases, thanks to their distributed architecture, deliver the sub-millisecond responsiveness and elastic scalability that digital communication applications require.

- **Fraud Detection:** For financial service organizations, fraud detection is essential to reducing profit loss, minimizing financial exposure and complying with regulations. When customers pay with a credit or debit card, they expect immediate confirmation. The process impacts both the enterprise and its customers. Fraud Detection relies on data – detection algorithm rules, customer information, transaction information, location, time of day and more –'applied at scale and in less than a millisecond. While relational databases struggle to meet this low latency requirement, elastically scalable NoSQL databases can reliably deliver the required performance.

Summary

- NoSQL describes the wide variety of database technologies created to address the shortcomings of RDBMS and the demands of modern software development.
- NoSQL database is used for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps.
- Using NoSQL database, different kinds of data like structured and semi structured data can be analyzed and analytical activities can be performed.
- There are 4 basic types of NoSQL databases: Key-Value Store , Document-based Store, Column-based Store, Graph based store.
- NoSQL databases are a more flexible, scalable, and less expensive alternative to relational databases.

Check Your Understanding

1. _____ stores are used to store information about networks, such as social connection.
   (a) Key-value (b) Wide-column
   (c) Document (d) Graph
2. NoSQL databases are used mainly for handling large volumes of _____ data.
   (a) unstructured (b) structured
   (c) semi-structured (d) All of the mentioned
3. Which of the following are the simplest NoSQL databases?
   (a) Key-value (b) Wide-column
   (c) Document (d) All of the mentioned
4. Point out the wrong statement.
   (a) Non Relational databases require that schemas be defined before you can add data.
   (b) NoSQL databases are built to allow the insertion of data without a predefined schema.

   (c) NewSQL databases are built to allow the insertion of data without a predefined schema.
   (d) All of the mentioned
5. Which of the following is not a NoSQL database?
   (a) MongoDB (b) Cassandra
   (c) SQL Server (d) None of the mentioned
6. Which of the following is not a feature for NoSQL databases?
   (a) Data can be easily held across multiple servers
   (b) Relational Data
   (c) Scalability
   (d) Faster data access than SQL databases
7. NoSQL systems are also referred to as _____.
   (a) "Not-On-SQL" (b) "N-Only-SQL"
   (c) "No-onlySQL" (d) "Not-Only
8. One of classified NoSQL databases is _____.
   (a) Key-value (b) value
   (c) key (d) DataNode
9. Most NoSQL databases support automatic _____.
   (a) processing (b) scalability
   (c) replication (d) reducing
10. "NoSQL is a Relational data Model": State true/false?
   (a) True (b) False

ANSWER KEY

1. (d)	2. (a)	3. (a)	4. (a)	5. (c)
6. (b)	7. (d)	8. (a)	9. (c)	10. (b)

Practice Questions

Q.I: Answers the following questions in short.

1. What is NoSQL database?
2. Which are types NoSQL?
3. Who invented the concept of NoSQL?
4. Name the companies which are mostly use NoSQL.
5. What is the use of Graph store?

Q.II: Answers the following questions.

1. List characteristics of NoSQL.
2. State advantages and disadvantages of NoSQL.
3. Explain in detail types of NoSQL.
4. Describe the need of NoSQL.
5. How does column-oriented NoSQL differ from document-oriented?

Q.III: Write a short note on:

1. Graph store
2. Column store
3. Key-value store
4. Document store
5. Use of NoSQL database in content management.