UNIT I INTRODUCTION TO DBMS

Syllabus:

Introduction - Database System Applications-Purpose of Database Systems- View of Data- Data Abstraction- Instances and Schemas- Relational Databases - Database Design- Data model- Integrity constraints -The Entity - Relationship Model - Transaction Management - Database Architecture - Data Storage and Querying - Database Users and Administrators.

DBMS stands for **D**ata**b**ase **M**anagement **S**ystem. We can break it like this DBMS = Database + Management System. Database is a collection of data and Management System is a set of programs to store and retrieve those data.

Definition:

DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

What is the need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization:

1. **Storage of data.**
2. **Retrieval of data**.

Storage:

- According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.
- Let's take a layman example to understand this:
  In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account.
- Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the

information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

Fast Retrieval of data:

- ❖ Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed.
- ❖ Database systems ensure that the data is retrieved as quickly as possible.

Applications of Database Management Systems:

- **Telecom**: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry**: Where it is a manufacturing unit, warehouse or distribution Centre, each one needs a database to keep the records of ins and outs. For example distribution Centre should keep a track of the product units that supplied into the Centre as well as the products that got delivered out from the distribution Centre on each day; this is where DBMS comes into picture.
- **Banking System**: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Education sector**: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping**: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

Drawbacks of File system:

- **Data Isolation**: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

- **Duplication of data** – Redundant data
- **Dependency on application programs** – Changing files would lead to change in application programs.

Advantage of DBMS over File System:

There are several advantages of Database management system over file system. Few of them are as follows:

1. No redundant data – Redundancy removed by data normalization
2. Data Consistency and Integrity – data normalization takes care of it too
3. Secure – Each user has a different set of access
4. Privacy – Limited access
5. Easy access to data
6. Easy recovery
7. Flexible

Disadvantages of DBMS:

1. DBMS implementation cost is high compared to the file system
2. Complexity: Database systems are complex to understand
3. Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

DATA MODELS IN DBMS

Data model - A collection of concepts that can be used to describe the structure of a database - provides the necessary means to achieve the abstraction.

A **Data Model** is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

It is a tool for data abstraction. A collection of tools for describing

- ❖ data
- ❖ data relationships
- ❖ data semantics
- ❖ data constraints

A data model is described by the schema, which is held in the data dictionary.

- Student(Stu_Id, Stu_Name, Stu_Add)
- Course(Course_Id, lecturer)

Categories of Data Models:

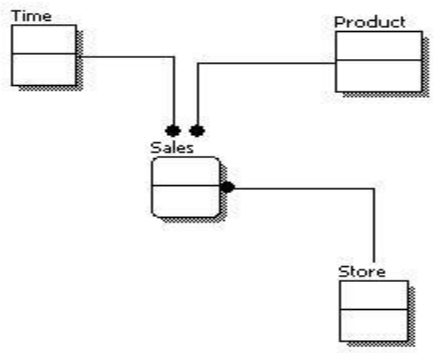
Many data models have been proposed, which can categories according to the types of concepts they use to describe the database structure.

1. High level or Conceptual data models.
   - Entity Relationship (E-R) model.
   - Object model
2. Low level or Physical data model.
3. Representational or Implementation data model.
   - Relational model.
   - Network model.
   - Hierarchical model.

High-level or **conceptual data models:**

A conceptual data model identifies the highest-level relationships between the different entities. Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.



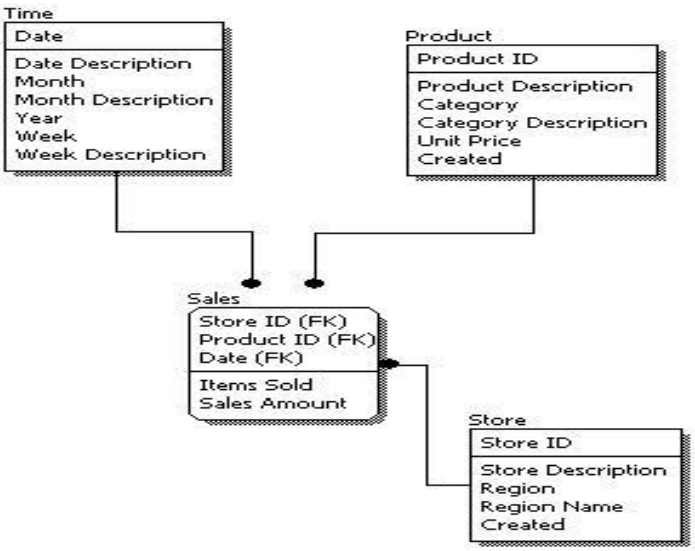
Physical (low-level) data models:

Physical data model represents how the model will be built in the database.

A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

Features of a physical data model include:

- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.



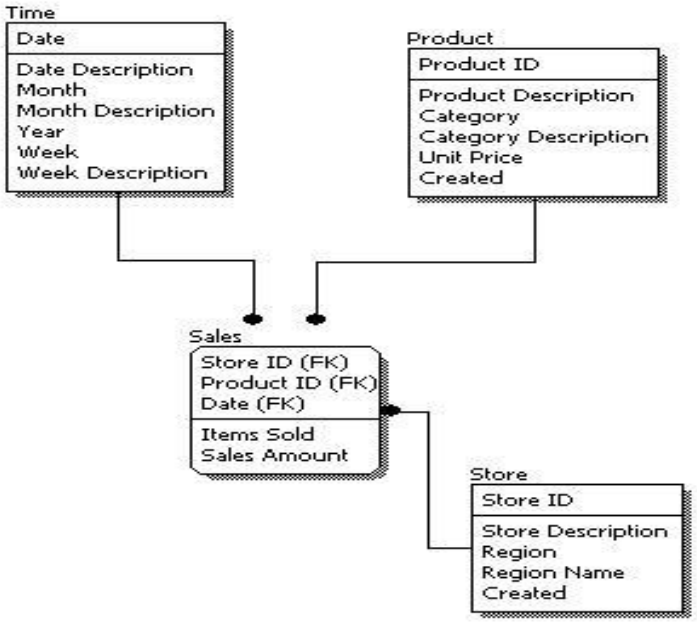
Implementation (Representational) data models:

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.



Conceptual, Logical, and Physical Data Models

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

INSTANCES AND SCHEMAS:

Schema: A logical structure of a database is called the schema. Schema is of three types.

1. Physical schema or internal schema
2. Logical schema
3. View schema or external schema

- The design of a database at physical level is called **physical schema**, 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 generally describes end user interaction with database systems.

Instance: The data stored in database at a particular moment of time is called instance of 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.

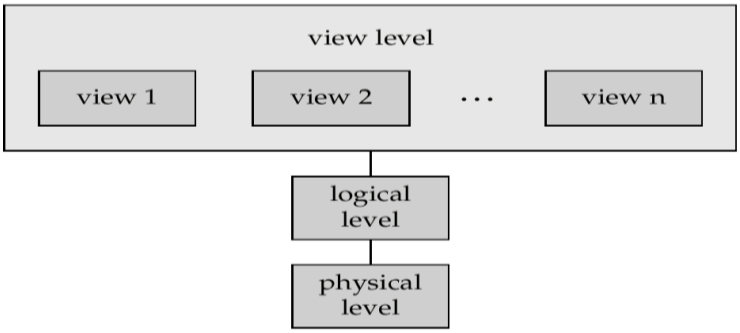
VIEWS OF DATA:

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction.

To understand the view of data, you must have a basic knowledge of data abstraction and instance & schema.

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.



Three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

Example: Let's say we are storing customer information in a customer table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

Data Independence:

Change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** - is the capacity to change the conceptual schema without having to change external schemas or application programs.
2. **Physical data independence** - is the capacity to change the internal schema without having to change conceptual schemas.

3. Applications depend on the logical schema. In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others

DBMS LANGUAGES:

Database languages are used for read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:

1. Data Definition Language (DDL): DDL is used for specifying the database schema. Let's take SQL for instance to categorize the statements that comes under DDL.

- To create the database instance – CREATE
- To alter the structure of database – **ALTER**
- To drop database instances – DROP
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**

All these commands specify or update the database schema that's why they come under Data Definition language.

2. Data Manipulation Language (DML): DML is used for accessing and manipulating data in a database.

- To read records from table(s) – SELECT
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – UPDATE
- Delete all the records from the table – DELETE

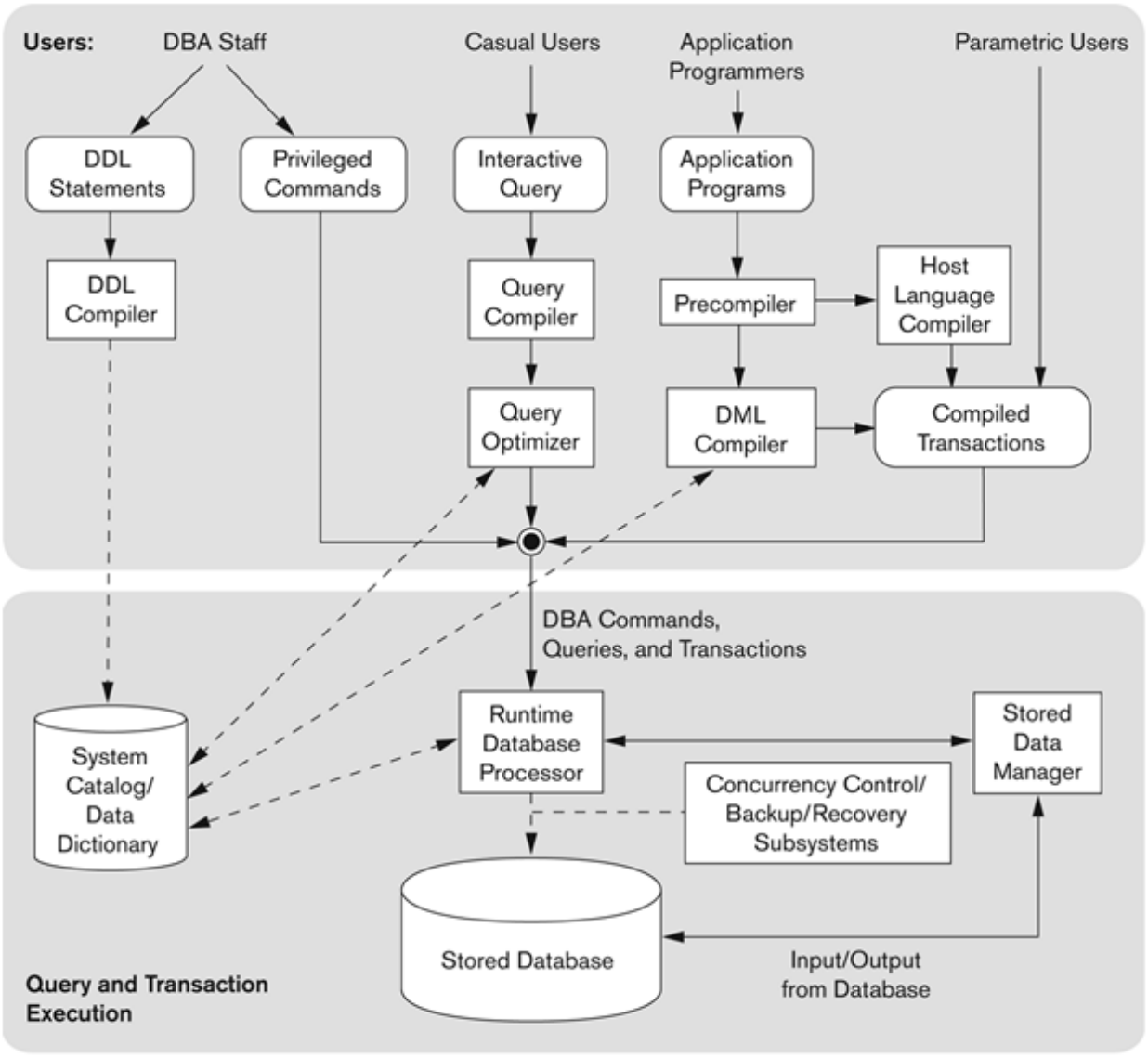
3. Data Control language (DCL): DCL is used for granting and revoking user access on a database –

- To grant access to user – GRANT
- To revoke access from user – REVOKE

In practical data definition language, data manipulation language and data control languages are not separate language; rather they are the parts of a single database language such as SQL.

4. **Transaction Control Language (TCL):** Transaction Control Language statements are used to control the transactions in the database.
    - SAVE POINT
    - ROLLBACK
    - COMMITE

DATABASE SYSTEM ARCHITECTURE



Components of DB Architecture

The DBA staff: Works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

Casual users: Those who work with interactive interfaces to formulate queries.

Application programmers: Those who create programs using some host programming languages

Parametric users: Who does data entry work by supplying parameters to predefined transactions?

The DDL compiler: Processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Query Compiler: Interactive Queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form.

The Pre-Compiler: Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a PreCompiler. It extracts DML commands from an application program written in a host programming language.

DML Compiler: DML commands are sent to the DML compiler for compilation into object code for database access

Host Language Compiler: The rest of the program which is extracted from precompiler is sent to the host language compiler

The query optimizer:

- It is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.
- It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Runtime Database Processor: The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

The System Catalog/Data Dictionary:

- The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.
- In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Stored data manager: Which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory?

Concurrency control / backup / recovery systems: They are integrated into the working of the runtime database processor for purposes of transaction management.

Stored Database: It is storing database in disk.

DATABASE USERS AND ADMINISTRATOR:

Database Users

Users are differentiated by the way they expect to interact with the system.

- **Application programmers**: interact with system through DML calls.
- **Sophisticated users** form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naive users** – invoke one of the permanent application programs that have been written previously

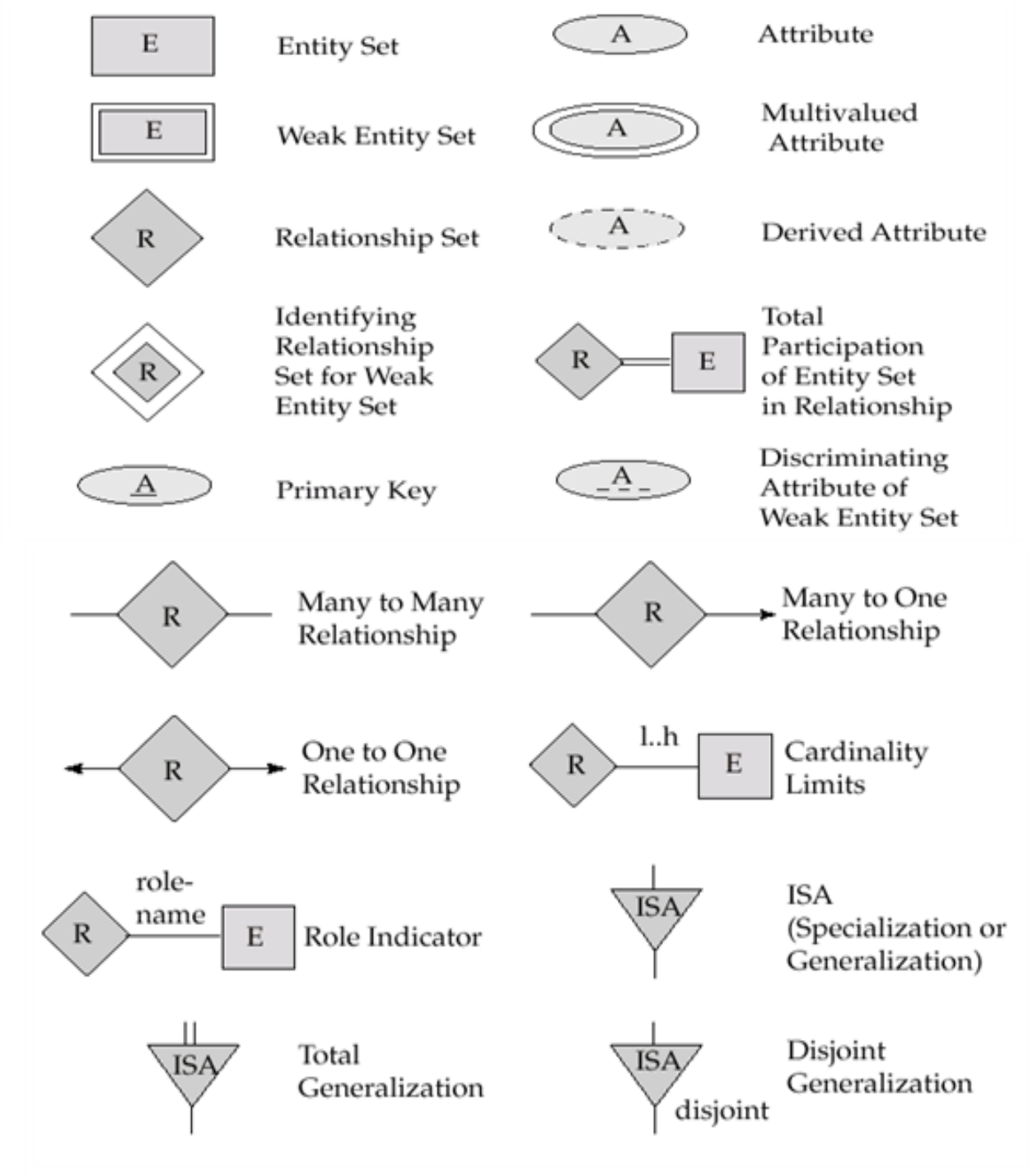
Database Administrator:

- Coordinates all the activities of the database system.
- The database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties (Roles) include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

E-R MODEL

An **Entity–Relationship Model (ER model)** is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.

Here are the geometric shapes and their meaning in an E-R Diagram –



Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

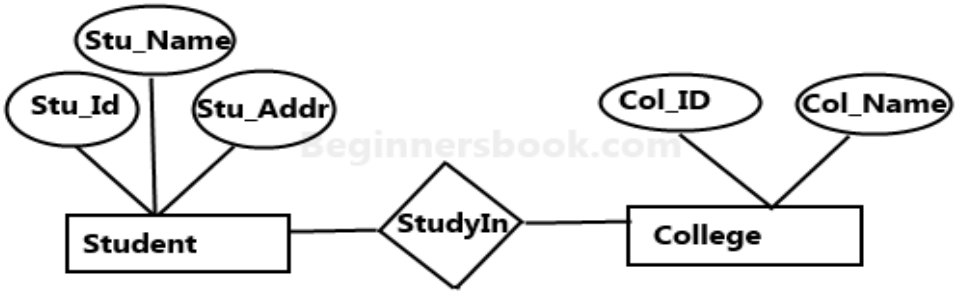
Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

A sample E-R Diagram:



Sample E-R Diagram

An **Entity** is represented by a set of attributes, which are descriptive properties possessed by all members of an entity set.

Attribute types:

- *Simple* and *composite* attributes.
- *Single-valued* attributes.
- *Multi-valued* attributes.

An attribute that can hold multiple values is known as multivalued attribute. We represent it with double ellipses in an E-R Diagram.

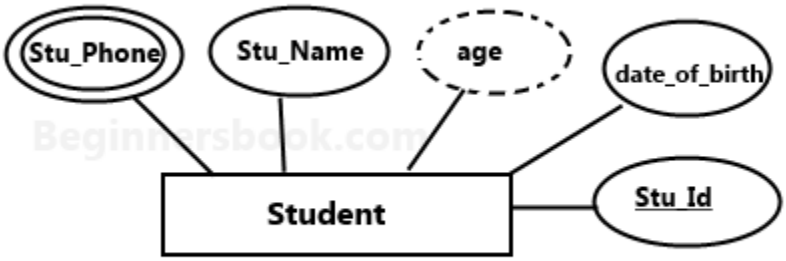
E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.

- *Derived* attributes:

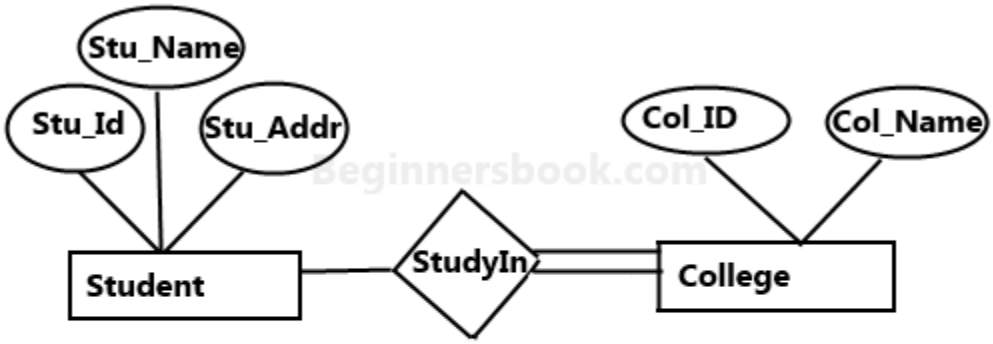
A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed ellipses in an E-R Diagram.

E.g. Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:

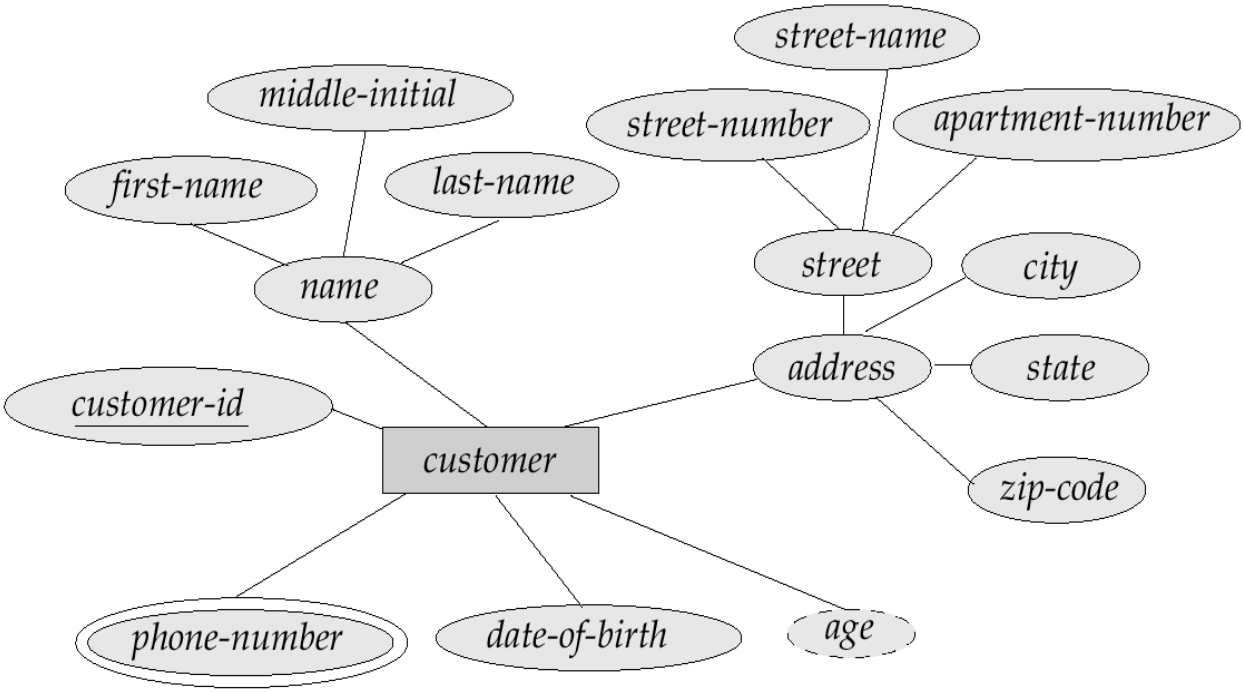


Total Participation of an Entity set:



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

E-R Diagram with Composite, Multivalued, and Derived Attributes

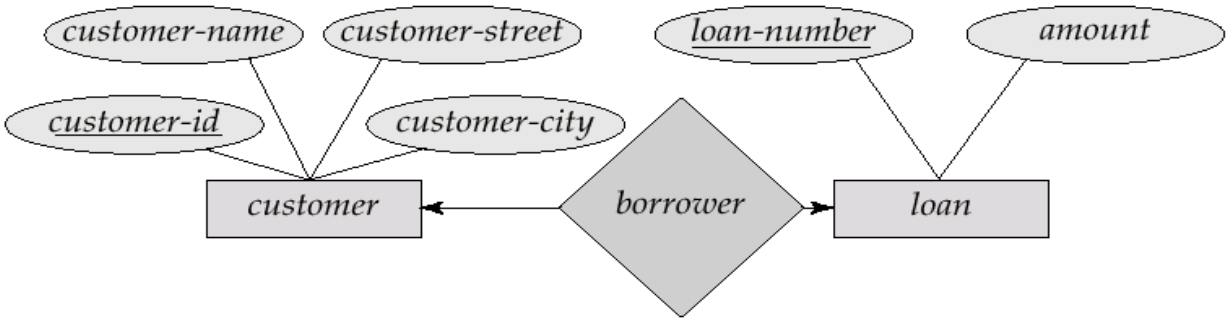


Cardinality Constraints:

We express ***cardinality constraints*** by drawing either a directed line (→), signifies "**one**" An undirected line (—), signifies "**many**" between the relationship set and the entity set.

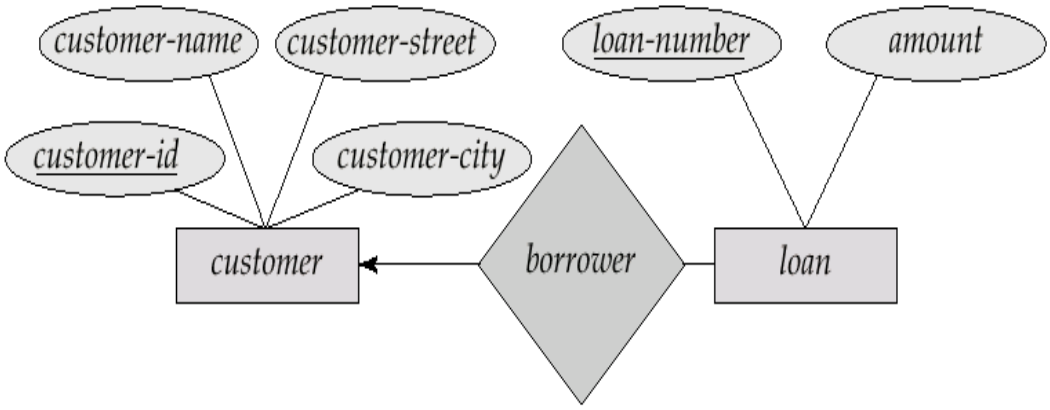
One-to-one relationship

- A customer is associated with at most one loan via the relationship *borrower*
- A loan is associated with at most one customer via *borrower*



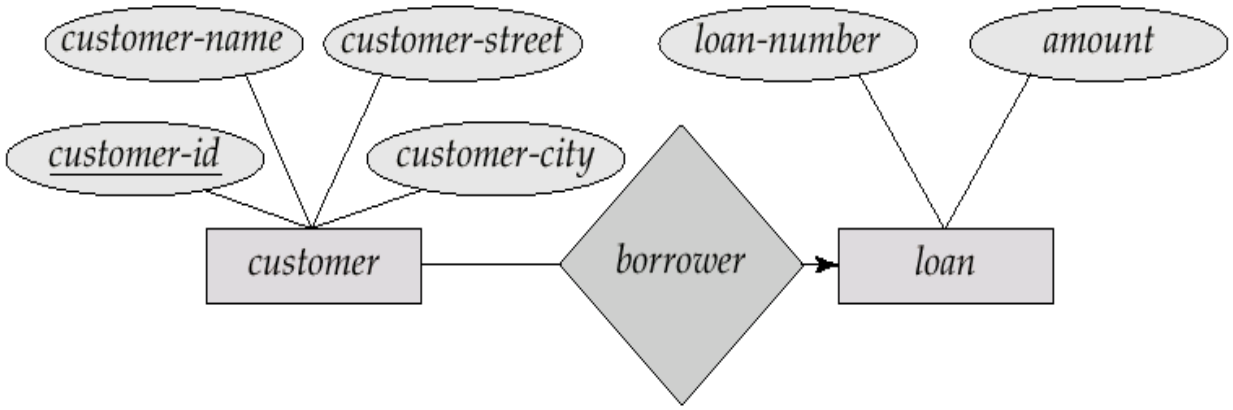
One-To-Many Relationship:

In the one-to-many relationship a loan is associated with at most one customer via *borrower*; a customer is associated with several loans via *borrower.*



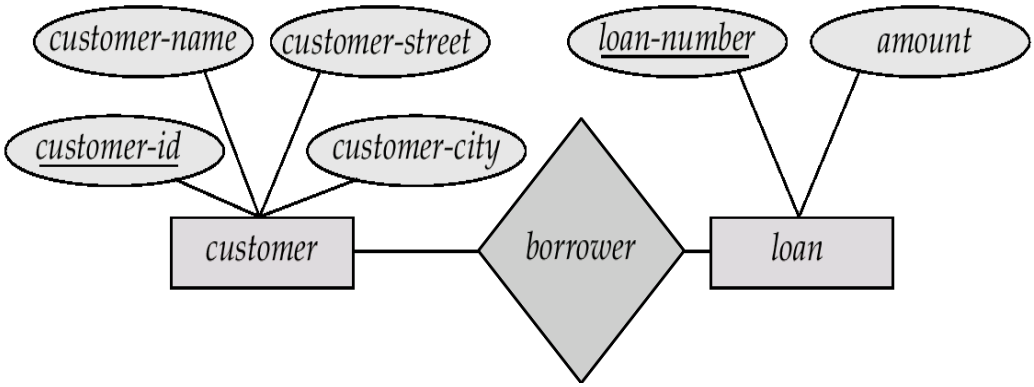
Many-To-One Relationships

Customers via *borrower*, a customer is associated with at most one loan via *borrower*



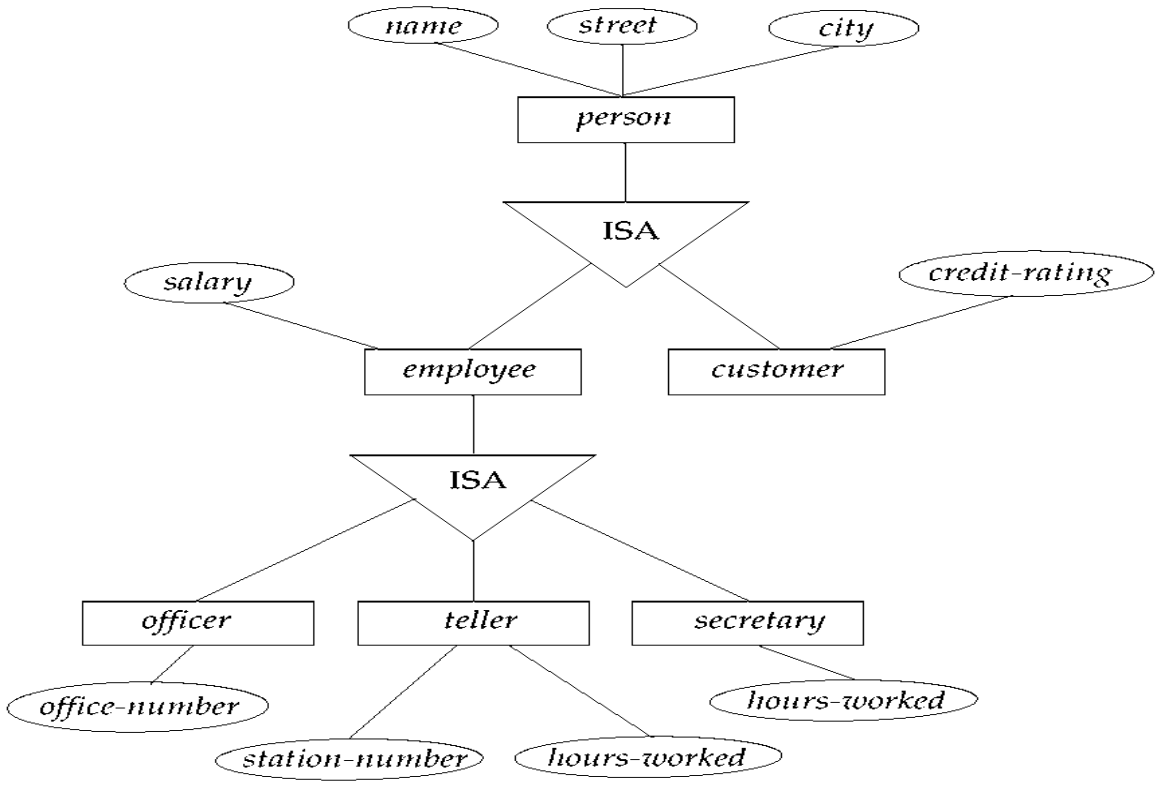
Many-To-Many Relationship

- A customer is associated with several loans via borrower
- A loan is associated with several customers via borrower



Specialization

- Top-down design process: We designate subgroupings within an entity set that are distinctive from other entities in the set.
- These sub groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (E.g. *customer* "is a" *person*).
- Attribute inheritance – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked



Generalization

- A bottom-up design process – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Relational model in DBMS

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

Sample relationship Model: Student table with 3 columns and four records.

Stu_Id	Stu_Name	Stu_Age
BTU10869	Bumesh	20
BTU10870	Pavan	20
BTU10864	Tharun	19
BTU10879	Arun	19

Course table: Course table

Stu_Id	Course_Id	Course_Name
BTU10869	Co1	Cloud Computing
BTU10870	Co2	OS
BTU10864	C22	DBMS
BTU10879	C39	Computer Networks

Here Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

RDBMS Concepts

RDBMS stands for relational database management system. A relational database has following major components: Table, Record / Tuple, Field & Column /Attribute.

Table:

A table is a collection of data represented in rows and columns. For e.g. following table stores the information of students.

Stu_Id	Stu_Name	Stu_Add	Stu_Age
BTU101	Lokeshwari	Dhayal bagh, Agra	20
BTU102	Bhavana	Delhi	20
BTU103	Saya	Gurgaon	20
BTU104	Bharathi	Chennai	20

Records / Tuple:

Each row of a table is known as record or it is also known as tuple. For e.g. the below row is a record.

BTU102	Bhavana	Delhi	20

Field:

The above table has four fields: Stu_Id, Stu_Name, Stu_Addr & Stu_Age.

Column / Attribute:

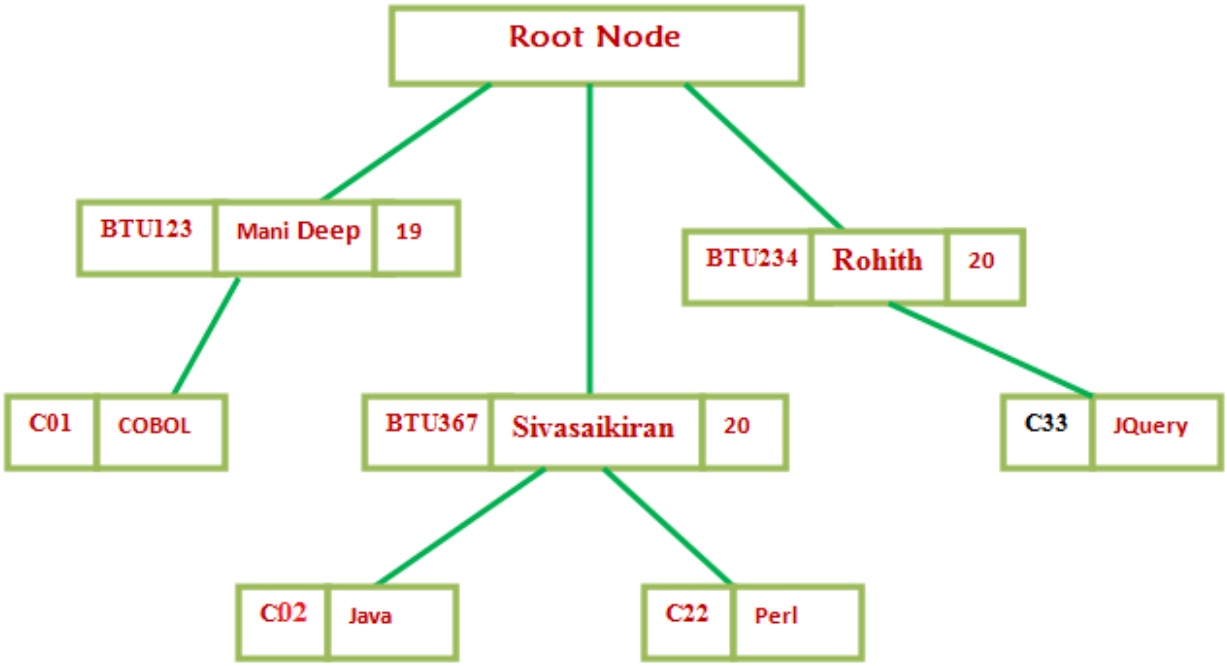
Each attribute and its values are known as attributes in a database. For e.g. Set of values of Stud_Id field is one of the four columns of the Student table.

Stu_Id
BTU101
BTU102
BTU103
BTU104

Hierarchical model in DBMS

In **hierarchical model**, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

Sample Hierarchical Model Diagram:



Example of hierarchical data represented as relational tables: The above hierarchical model can be represented as relational tables like this:

Stu_Id	Stu_Name	Stu_Age
BTU123	Mani Deep	19
BTU367	Sivasaikirann	20
BTU234	Rohith	20

Course Table:

Course_Id	Course_Name	Stu_Id
C01	Cobol	BTU123
C02	Java	BTU367
C21	Perl	BTU367
C33	JQuery	BTU234

UNIT 2: RELATIONAL DATABASES

Syllabus:

Introduction to Relational model - Structure of Relational Databases - Relational database design - keys - Database Schema and Schema Diagrams - Relational Query Languages - Relational algebra and Relational Operations - Relational calculus -Introduction to SQL - Overview of the SQL Query Language - Basic Structure of SQL Queries - Functional dependency -Normal Forms.

RELATIONALDATAMODEL

- The relational model was introduced by **Dr. E.F Codd**in **1970**.
- The relation at model represents data in the form of two-dimensional tables. Each table represents some real-world entity or thing.

Characteristics of Relational Model

- This model data in the database as simple **row/column**.
- Each table is an **independent entity** and there is no physical relationship between tables.
- Relational model of data management is based on **set theory**.
- The user interface used in relational models in non-procedural because only **what needs to be done** is specified and **not how it has to be done**.

E.F. CODD'S LAWS

1. **Information representation**:

   All information stored in relational database is represented only by data item values, which are stored in tables.

2. **Logical accessibility**:

   Every data item value stored in a relational database is accessible by stating the name of the table it is stored in, the name of the column stored and the primary key that defines the row in which the data stored

3. **Representation of null values**:

   The DBMS has a consistent method for representing null values.

4. **Catalog Facilities**:

The logical description of RDBMS is represented as same manner of ordinary data, this is done so that facilities of RDBMS itself can be used to maintain database description.

5. Data Language:

A RDBMS may support many types of languages for describing data and accessing the database.

6. View Updatability:

Any view can be defined using combination of base tables that are theoretically updatable, is capable of being updated by RDBMS.

7. Insert, update and delete:

Any operand that describes the result of a single retrieval operation is capable of being applied to an insert, update and delete.

8. Physical data independence:

Changes made to physical storage or access methods do not require changes in application program.

9. Logical data independence:

Changes made to tables or do not require changes in application program.

10. Integrity constraints:

Constraints that apply to entity integrity and referential integrity are specifiable by the data language implemented by the DBMS and not by the statements coded into the application program.

11. Database Distribution:

The database language implemented by the RDBMS supports the ability to distribute the database without requiring the changes to be made to the application program.

12. Non-Subversion:

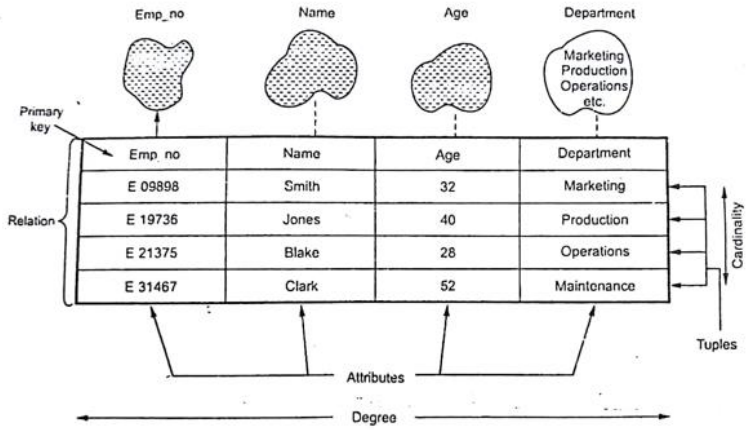
If the RDBMS supports facilities that allow application programs to operate on tables a row at a time, an application program using this type of database access is prevented from bypassing entity integrity or referential integrity constraints.

PRINCIPLE COMPONENTS OF RELATIONAL MODEL

1. Data Structure
2. Data integrity
3. Data Manipulation

1. Relational Data Structure

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Stu_Rollno, Stu_Name,etc.
2. **Tables** – In the Relational data model, the relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. ***Rows represent records*** and ***columns represent attributes***.
3. **Tuple** – It is nothing but a ***single row of a table***, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system.
8. Pool of value is **Domain**



Formal relational term	Informal equivalence
relation	table
tuple	row or record
cardinality	number of rows
attribute	column or field
degree	number of columns
primary key	unique identifier
domain	pool of legal values

2. Date Integrity:

Integrity constraints provide a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.

Following are types of integrity constraints -

1. Domain Constraints

2. Referential integrity
3. Nulls
4. Entity Integrity
5. Enterprise constraints

❖ **Domain Constraints:** Domain constraints specify the set of values that can be associated with an attribute. They are tested easily by the system whenever a new data item is entered into the database

❖ **Referential Integrity:** A value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This is called as referential integrity.

❖ **Nulls:** Null represents a value for an attribute that is currently unknown or is not applicable for this tuple.

❖ **Entity Integrity:** In base relation no attribute of primary key can be null. A primary key is used to identify the tuple uniquely.

❖ **Enterprise Constraints:** These are additional rules specified by the users or administrators.

3. Data Manipulation:

The manipulative part of relational model consists of a set of operators known collectively as the ***relational algebra*** together with ***relational calculus***.

Advantages of relational model

- **Simplicity:** A relational data model is simpler than the hierarchical and network model.
- **Structural Independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use:** The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability:** It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence:** The structure of a database can be changed without having to change any application.
- **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of relational model

- Few relational databases have ***limits on field lengths*** which can't be exceeded.
- Relational databases can ***sometimes become complex*** as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems ***may lead to isolated databases*** where the information cannot be shared from one system to another.

KEYS:

What are Keys?

- A DBMS key is an attribute or set of an attribute which helps you to **identify a row (tuple*)*** in a relation (table).
- They allow you to **find the relation between two tables**.
- Keys help you **uniquely identify a row** in a table by a combination of one or more columns in that table.

Example:

STUD_ID	STUD_NAME	EMAIL_ID
U18CN167	Manideep	u18cn167@biher.edu.in
U18CN187	Vishal	u18cn187@biher.edu.in
U18CN189	Lokeshwari	u18cn189@biher.edu.in
U18CN200	Pavan Satya	u18cn200@biher.edu.in

In the above-given example, **STUD_ID is a primary key** because it uniquely identifies a student record. In this table, no other student can have the same STUD_ID.

Why we need a Key?

- Keys help you to identify any row of data in a table.
- Keys allows you to establish a relationship between and identify the relation between tables
- Keys help you to enforce identity and integrity in the relationship.

Various Keys in Database Management System

DBMS has flowing seven types of Keys each have their different functionality:

1. Super Key
2. Primary Key

3. Candidate Key
4. Alternate Key
5. Foreign Key
6. Compound Key
7. Composite Key
8. Surrogate Key

1) Primary Key:

- A column or group of columns in a table which helps us to ***uniquely identify every row*** in the table is called a primary key.
- The same value ***can't appear more than once*** in the table.
- ***Rules*** for defining Primary key:
    1. The primary key field ***cannot be null***.
    2. The value in a primary key column ***can never be modified or updated*** if any foreign key refers to that primary key.

STUD_ID	STUD _NAME	EMAIL_ID
U18CN167	Manideep	u18cn167@biher.edu.in
U18CN187	Vishal	u18cn187@biher.edu.in
U18CN189	Lokeshwari	u18cn189@biher.edu.in
U18CN200	Pavan Satya	u18cn200@biher.edu.in

- From the above example, STUD_ID is a primary key attribute.

2) Candidate Key:

- ***"A candidate key is an attribute or set of an attribute which can uniquely identify a tuple***."
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

Example:

- In the STUDENT table, STUD_ID is best suited for the primary key.
- Rest of the attributes like REG_NO, and EMAIL_ID are considered as candidate keys which help us to uniquely identify the student record in the table.

STUD_ID	REG_NO	STUD _NAME	EMAIL_ID
U18CN167	1XXX101	Manideep	u18cn167@biher.edu.in
U18CN187	1XXX102	Vishal	u18cn187@biher.edu.in
U18CN189	1XXX103	Lokeshwari	u18cn189@biher.edu.in
U18CN200	1XXX112	Pavan Satya	u18cn200@biher.edu.in

3) Super Key:

- ***"Super key is a set of an attribute which can uniquely identify a tuple***."
- Super key is a superset of a candidate key.

Example: In the above STUDENT table, for (**STUD_ID, STUD_NAME**) the name of two students can be the same, but their **STUD_ID** can't be the same.

Hence, this combination can also be a key.

- The super key would be **STUD_ID**, (**STUD_ID, STUD _NAME**), etc.

4) Alternate Keys:

- ***"Alternate Keys is a column or group of columns in a table that uniquely identify every row in that table."***
- A table can have multiple choices for a primary key but only one can be set as the primary key. All the ***keys which are not primary key are called an Alternate Key***.

Example:

- In this table, STUD_ID, REG_NO, EMAIL_ID are qualified to become a primary key. But since STUD_ID is the primary key, ***REG_NO, EMAIL_ID becomes the alternative key***.

STUD_ID	REG_NO	STUD_NAME	EMAIL_ID
U18CN167	1XXX101	Manideep	u18cn167@biher.edu.in
U18CN187	1XXX102	Vishal	u18cn187@biher.edu.in
U18CN189	1XXX103	Lokeshwari	u18cn189@biher.edu.in
U18CN200	1XXX112	Pavan Satya	u18cn200@biher.edu.in

5) Foreign Key:

- ***"Foreign Key is a column that creates a relationship between two tables."***
- The purpose of Foreign Keys is to ***maintain data integrity*** and allow ***navigation between two different instances*** of an entity.

- It acts as a ***cross-reference between two tables*** as it references the primary key of another table.

Table Name: Faculty

Faculty_Id	Fac_Name	Dept_Code	Email_Id
BU2702	Nithiyanandam	BTUCS51	nithiyanandam@biher.edu.in
BU2703	Janagiraman	BTUCS51	janagiraman@biher.edu.in
BU2543	Elankavi	BTUCE56	elankavi@biher.edu.in
BU2427	Sivaranjani	BTUCS51	ranjibala@gmail.com
BU10270	Arivuselvi	BYUEC55	Arivuselvi270@yahoo.com

Table Name: Department

Dept_Code	Dept_Name
BTUCS51	CSE
BTUCE56	CIVIL
BTUEC55	ECE
BTUEE58	EEE
BTUME60	MECH

6) Compound Key:

- ***"Compound Key has two or more attributes that allow you to uniquely recognize a specific record. "***
- It is possible that each column may not be unique by itself within the database.
- However, when combined with the other column or columns the ***combination of composite keys become unique***.
- The purpose of ***compound key is to uniquely identify each record*** in the table.

RELATIONAL ALGEBRA

- Relational algebra presents the ***basic set of operations*** for relational model.
- It is a ***procedural language***, which describes the procedure to obtain the result.
- Relational algebra is prescriptive because it describes the **order of operations** in the query that specifies ***how*** to retrieve the result of a query.

Relational Algebra Operation:

a) **Selection**
b) **Projection**
c) **Rename**
d) **Union**
e) **Intersection**
f) **Set difference**
g) **Cartesian product**
h) **Join**
i) **division**
j) **Aggregate function**
k) **Outer Join**

a) Select Operation:

- The select operation selects the tuple that satisfy a given predicate (condition).
- The select operation is represented as follows

  $\sigma_{\text{<Condition>}}$ **(Relation)**

  σ - Denote the select operator

  Condition – Denotes **expression** specified on the attributes of the **Relation** (Table).

Table Name: **BOOK_DETAIL**

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

Select Condition:

- Operators: <, ≤, ≥, >, =, ≠
- Simple selection condition:

  <attribute>operator<constant>

  <attribute>operator<attribute>

  <condition> AND <condition>

  <condition> OR <condition>

  NOT <condition>

Example:

1) Display the book detail published in the year 2000

$\sigma_{year=2000}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book4	Concrete Maths	Donald Knuth		2000	500

2) Display the books detail having price greater than 300.

$\sigma_{price>300}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

3) Display the books details whose publishing year is 2000 and the price is greater than 300.

$\sigma_{year=200 \text{ AND } price>300}$ (Book_Detail)

B_ID	TITTLE	AUTHOR	PUBLISHER	YEAR	PRICE
Book4	Concrete Maths	Donald Knuth		2000	500

b) <u>Project Operation</u>:

The projection operation ***selects certain attributes*** from the relation while discarding others.

The projection operation is represented as follows

$\Pi_{attribute\ list}(Relation)$

Table Name: BOOK_DETAIL

B_ID	TITLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	RamezElmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350
B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald knuth		2000	500

Examples:

1) Display all book titles with author name

$\Pi_{title, author}$ (Book_detail)

TITLE	AUTHOR
DBMS	Ramez Elmarsi
Java Programming	K Somasundaram
Computer Network	S Tanenbaum
Concrete Maths	Donald Knuth

Composition of Select and Project Operation:

The relational operations select and project can be combined to form a complicated query.

Example:

a) Display the Book titles having price greater than 300.

$\Pi_{title, price}(\sigma_{price>300}$ (Book_Detail)

TITLE	PRICE
Java Programming	350
Computer Network	450
Concrete Maths	500

c) <u>Rename Operation</u>:

- In the relational algebra, you can rename either the *relation* or *the attributes* or *both*.
- The symbol $\boldsymbol{\rho}$ (rho) is used to denote the rename operator.
  1. $\boldsymbol{\rho}$ S (new attribute names) (R)
  2. $\boldsymbol{\rho}$ S (R)
  3. $\boldsymbol{\rho}$ (new attribute names) (R)

'**S**' is the new relation and '**R**' is original relation.

Table Name: BOOK_DETAIL

B_ID	TITLE	AUTHOR	PUBLISHER	YEAR	PRICE
B0001	DBMS	Ramez Elmarsi	McGraw-Hill	2000	250
Book2	Java Programming	K Somasundaram	Jaico Publishing	2004	350

B00k3	Computer Network	S Tanenbaum	Pearson	2003	450
Book4	Concrete Maths	Donald Knuth		2000	500

The Rename operator used on Book_Detail relation as follows

1) $\rho_{Book(bk\_no, bk\_title, bk\_author, bk\_publisher, pub\_year, b\_price)}$ (Book_Detail)

   - Here Relation name and attribute names are renamed.

2) ρ_{Book} (Book_Detail)

   - Here Relation name is renamed.

3) $\rho_{(bk\_no, bk\_title, bk\_author, bk\_publisher, pub\_year, b\_price)}$(Book_Detail)

   - Here attribute names are renamed.

SET OPERATION:

Union, intersection and **difference -** These operations require that the tables involved be ***union compatible***. Two relations are said to be union compatible if the following conditions are satisfied.

1. The two relations must contain the ***same number of columns***
2. Each column of the first relation must be ***either same data type*** or ***convertible to the same data type*** as corresponding column of the second relation

d) <u>Union Operation</u>:

- The operation is denoted by ***Depositor ∪ Borrower*** that includes all the tuples are either in depositor or borrower.
- Duplicates are eliminated.

Table Name: DEPOSITOR

CUST_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331002	K.BALACHANDAR	AVADI
5432331003	MADINA SUSHMA	T NAGAR
5432331003	PRATHYUSHA	T NAGAR
5432331005	NITISH	AMBATTUR
5432331006	SAI KRISHNA	GUINDY
5432331007	P.DIVYA KRISHNA	PAADI
5432331008	G.REETHIKA	AMBATTUR
5432331009	PADMINI	PORUR
5432331010	SAI MAHESH	GUINDY

Table Name: BORROWER

BR_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331014	VINAY	PORUR
5432331003	MADINA SUSHMA	T NAGAR
5432331078	ARIJITH	ANNA NAGAR
5432331089	DILEEP	ANNA NAGAR
5432331008	G.REETHIKA	AMBATTUR
5432331010	SAI MAHESH	GUINDY

Table Name: DEPOSITOR

CUST_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331002	K. BALACHANDAR	AVADI
5432331003	MADINA SUSHMA	T NAGAR
5432331003	PRATHYUSHA	T NAGAR
5432331005	NITISH	AMBATTUR
5432331006	SAI KRISHNA	GUINDY
5432331007	P. DIVYA KRISHNA	PAADI
5432331008	G. REETHIKA	AMBATTUR
5432331009	PADMINI	PORUR
5432331010	SAI MAHESH	GUINDY

Table Name: BORROWER

BR_ID	CUST_NAME	CITY
5432331001	SAI BHARATH	AVADI
5432331014	VINAY	PORUR
5432331003	MADINA SUSHMA	T NAGAR
5432331078	ARIJITH	ANNA NAGAR
5432331089	DILEEP	ANNA NAGAR
5432331008	G. REETHIKA	AMBATTUR
5432331010	SAI MAHESH	GUINDY

DEPOSITOR U BORROWER

CUST_NAME	CITY
REETHIKA	AMBATHUR
ARIJITH	ANNA NAGAR
DILEEP	ANNA NAGAR
K. BALACHANDAR	AVADI
MADINA SUSHMA	T NAGAR
NITISH	AMBATHUR
P. DIVYA KRISHNA	PAADI
PADHMINI	PORUR
PRATHYUSHA	T NAGAR
REETHIKA	AMBATHUR
SAI BHARATH	AVADI
SAI BHARATH	AVADI
SAI KRISHNA	GUINDY
SAI MAHESH	GUINDY
VINAY	PORUR

e) Intersection Operation:

- It includes all the tuples that are present in both depositor and borrower relations.
- The operation is denoted by ***Depositor*** ∩ ***Borrower.***

DEPOSITOR ∩ BORROWER

CUST_NAME	CITY
HARISH	GUINDY
NIKITHA P	AVADI
PADMINI	AMBATTUR
VISHNU TEJA	T NAGAR

f) Difference Operation:

- The difference operation is denoted by ***Depositor – Borrower***.
- It results the relation that contains all tuples in depositor but not in borrower.

DEPOSITOR – **BORROWER**

CUST_NAME	CITY
CHANDANA PRIYA	AVADI
GOPALAM KESAVA	PAADI
MOHITH	AMBATTUR
NACHIKETA KUMAR	PORUR
YASHWANTH	T NAGAR
YUVRAJ	GUINDY

BORROWER – DEPOSITOR

CUST_NAME	CITY
ARYAN	ANNA NAGAR
KUSHAL	PORUR
VASUJIT	ANNA NAGAR

g) Cartesian Product:

- The Cartesian product is also known as ***Cross Product*** or ***Cross Joins***, it is denoted by '**X**'.
- The Cartesian product of two relation A and B is denoted as **A X B**.

- When a ***join condition is omitted*** when getting result from two tables then that kind of query gives us Cartesian product, in which all combination of rows is displayed.
- All rows in the first table is joined to all rows of second table.
  - ✓ The result of Cartesian product of two relation which have **m** and **n columns** is a relation that has **m + n columns**.
  - ✓ Hence if the relation will have **x and y** tuples respectively, then the Cartesian product will have **n * m tuples**.

Table Name: **AUTHOR**

AUTHOR_ID	AUTHOR_NAME
125001	Ramez Elmarsi
125010	K Somasundaram
125017	S Tanenbaum
125020	Donald Knuth

Table Name:**BOOK_DETAIL**

BOOK_ID	TITLE
2578	DATABASE SYSTEMS
2568	JAVA PROGRAMMING
2588	COMPUTER NETWORKS
2548	CONCRETE MATHEMATICS

AUTHOR x BOOK_DETAIL

AUTHOR_ID	AUTHOR_NAME	BOOK_ID	BOOK_NAME
125001	Ramez Elmarsi	2578	database systems
125001	Ramez Elmarsi	2568	Java programming
125001	Ramez Elmarsi	2588	Computer networks
125001	Ramez Elmarsi	2548	Concrete mathematics
125010	k somasundaram	2578	database systems
125010	k somasundaram	2568	Java programming
125010	k somasundaram	2588	Computer networks
125010	k somasundaram	2548	Concrete mathematics
125017	S Tanenbaum	2578	database systems
125017	S Tanenbaum	2568	Java programming
125017	S Tanenbaum	2588	Computer networks
125017	S Tanenbaum	2548	Concrete mathematics
125020	Donald knuth	2578	database systems
125020	Donald knuth	2568	Java programming
125020	Donald knuth	2588	Computer networks
125020	Donald knuth	2548	Concrete mathematics

h) Natural Join:

- The natural join is a ***binary operation*** that allow us to ***combine certain selections and a Cartesian product*** into one operation.
- It is denoted by the join symbol ⋈.
- ***Duplicate columns eliminated*** from the result.

Table Name: **EMPLOYEE**

EMP_ID	EMP_NAME
E6501	BIKRAM SHAH
E6502	SOURAV KUMAR
E6503	RASHEED ALI
E6504	AJIT KUMAR

Table Name: **SALARY**

EMP_ID	SALRY
E6501	45000
E6502	43000
E6503	45000
E6504	44000

Example: Display the names of all employees with salary, we need to combine the employee and salary relation using ***Cartesian product without duplicated tuples***

$\Pi_{emp\_name,\ salry}(\sigma_{employee.emp\_id\ =\ salary.emp\_id}(\text{Employee X Salary}))$

The above query is rewritten using natural join as follows

$\Pi_{emp\_name,\ salry}(\text{Employee} \bowtie \text{Salary})$

EMP_NAME	SALARY
BIKRAM SHAH	45000
SOURAV KUMAR	43000
RASHEED ALI	45000
AJIT KUMAR	44000

i) Division Operation:

The division operation is denoted by **'÷'.** It is suited to queries that include the phrase for all.

Table Name: **ACCOUNT**

ACCT_NO	B_NAME	BALANCE
A-101	AVADI	25000
A-102	AMBATTUR	22000
A-203	GANDHI ROAD	15000
A-208	NEHRU PARK	42000
A-107	MAHINDRA CITY	53000
A-105	MARKET ROAD	15000
A-109	MARKET ROAD	26000

Table Name: **CUSTOMER**

ACCT_NO	CUST_NAME
A-101	SAI KRISHNA
A-102	CHALAMAIAH
A-203	SHATADRU
A-207	SANJAY DAS
A-107	MANISH KUMAR
A-105	BEJAWADA RAJA

Table Name: **BRANCH**

B_NAME	B_CITY	ASSETS
AVADI	CHENNAI	71000
AMBATTUR	CHENNAI	25000
GANDHI ROAD	KANCHIPURAM	34500
NEHRU PARK	THIRUVALLUR	27800
MAHINDRA CITY	CHENGLEPET	64000
MARKET	VELLORE	29500

Step:1 We can obtain all branches in Chennai by the expression

R1 = $\Pi_{b\_name}(\sigma_{b\_city = chennai}$ (Branch))

B_NAME
AVADI
AMBATTUR

Step: 2 We can find all (Cust_name, B_name) pairs for which the customer has an account at a branch by as follows

R2 = $\Pi_{cust\_name,\ b\_name}$(Customer ⋈ Account)

CUST_NAME	B_NAME
SAI KRISHNA	AVADI
CHALAMAIAH	AMBATTUR
SHATADRU	GANDHI ROAD
MANISH KUMAR	MAHINDRA CITY
BEJAWADA RAJA	MARKET ROAD

Step: 3 Now we need to find customers who appear in R2 with every branch name in R1.

$\Pi_{cust\_name,\ b\_name}$(Customer ⋈ Account)÷$\Pi_{b\_name}(\sigma_{b\_city = chennai}$ (Branch))

(or)

R2 ÷ R1

CUST_NAME
SAI KRISHNA
CHALAMAIAH

j) Aggregate Functions:

Aggregation function takes a collection of values and returns a single value as a result.

- **avg**: Average value
- **min**: Minimum value
- **max**: Maximum value
- **sum**: Sum of values
- **count**: Number of values

Table Name: **EMPLOYEE**

EMP_ID	EMP_NAME	DEPARTMENT	SALARY
1500101	SHARATH KUMAR	ACCOUNTS	45000
1500103	MANTHRI KARTHIK	SALES	43000
1500104	PALLI SWATHI	SALES	45000
1500105	DHANUSH KUMAR	IT	44000
1500108	HARINATH	IT	36000

Example:1 Calculate Total Salary paid to Employee

$G_{sum(salary)}$**(Employee)**

G – Signifies that aggregate function is to be applied, and its subscript specifies the aggregate operation to be applied.

SUM(SALARY)
213000

Example: 2 Display average salary of employees

$G_{avg(salary)}$**(Employee)**

AVG(SALARY)
42600

Example: 3 Count number of departments in Employee relation

$G_{count\text{-}distinct(department)}$**(Employee)**

DEPARTMENT
3

Example: 4 Display total salary of employees as department wise

Department$G_{sum(salary)}$(Employee)

DEPARTMENT	SUM(SALARY)
ACCOUNTS	45000
SALES	88000
IT	80000

k) Outer Join:

- The outer join operation is an extension of the **JOIN** operation to deal with missing information.

Table Name: **EMPLOYEE**

EMP_NAME	CITY
VIJAYA BHASKAR	PUNE
THUNGA PRATAP	MUMBAI
MARUTHI KUMAR	NASHIK
ROSHAN KUMAR	SOLAPUR

Table Name: **SALARY**

EMP_NAME	DEPARTMENT	SALARY
VIJAYA BHASKAR	PLANNING	65000
THUNGA PRATAP	ANALYST	72000
MANOHARA CHARY	PLANNING	64000
ROSHAN KUMAR	ANALYST	65000

EMPLOYEE⋈ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

- ✓ ***Maruthi Kumar Data is lost***
- ✓ ***Manohara Chary Data is lost.***

To avoid this loss of information, we can use outer join

Types of outer join:

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

1. **Left Outer Join**:
   - ❖ Denoted by ⟕
   - ❖ It takes ***all tuples in the left relation*** that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.

EMPLOYEE ⟕ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MARUTHI KUMAR	NASHIK		
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

2. **Right Outer Join:**
   - ❖ The Right outer join is denoted by ⟖
   - ❖ It pads ***tuples from right relation*** that did not match with the tuple from the left relation with nulls and adds them to the result of the natural join.

EMPLOYEE ⟖ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MANOHARA CHARY		PLANNING	64000
ROSHAN KUMAR	SOLAPUR	ANALYST	65000

3. **Full Outer Join:**
   - ❖ Full outer join denoted by⟗
   - ❖ It pads ***tuples from left relation*** that did not match with right relation and as well as ***tuples from right relation*** that did not match with any tuple from left relation and adds them to the resultant relation.

EMPLOYEE ⋉ SALARY

EMP_NAME	CITY	DEPARTMENT	SALARY
VIJAYA BHASKAR	PUNE	PLANNING	65000
THUNGA PRATAP	MUMBAI	ANALYST	72000
MARUTHI KUMAR	NASHIK		
ROSHAN KUMAR	SOLAPUR	ANALYST	65000
MANOHARA CHARY		PLANNING	64000

RELATIONAL CALCULAS:

- Relational Calculus is a higher-level ***Declarative language***.
- Relational calculus defines ***what result is to be obtained***.
- Relational Calculus ***does not specify the sequence of operations*** in which query will be evaluated.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

(a). Tuple Relational Calculus (TRC)

- The Tuple Relational Calculus **list the tuples** to be selected from a relation, based on a certain **condition** provided.
- It is formally denoted as:
- ***{ t | P(t) }***
- Where **t** is the set of tuples from which the condition **P** is true.

(b). Domain Relational Calculus (DRC)

- Domain Relational Calculus **list the attributes** to be selected from a relation, based on certain **condition**.
- The formal definition of Domain Relational Calculus is as follow:

  {<X1, X2, X3, . . .Xn> | P(X1, X2, X3, . . . Xn)}

  Where **X1, X2, X3, . . .Xn** are the attributes and **P** is the certain condition.

Difference between Relational Algebra and Relational Calculus

BASIS	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Calculus is Declarative language.
State	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Calculus can be domain dependent.

STRUCTURE QUERY LANGUAGE (SQL):

SQL (Structured Query Language) is a computer language aimed to store, manipulate, and query data stored in relational databases.

TYPES OF SQL COMMANDS

SQL is used to control all of the functions that a DBMS provides for its users, including:

- ***Data definition***, SQL lets a user define the structure and organization of the stored data and relationship among the stored data items.
- ***Data retrieval***, SQL allows a user or an application program to retrieve stored data from the database and use it.
- ***Data manipulation***, SQL allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data.
- ***Access control***, SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- ***Data sharing***, SQL is used to coordinate data sharing by concurrent users, ensuring that they do not interface with one another.
- ***Data integrity***, SQL defines integrity constraints in the database, protecting it from corruption due to inconsistent updates or system failures.

DATA DEFINITION LANGUAGES (DDL)

A Data Definition Language (DDL) statement are used to define the database structure or schema.

DDL commands:

- ✓ CREATE
- ✓ ALTER
- ✓ DROP
- ✓ RENAME
- ✓ TRUNCATE

Create Table:

To make a new database, table, index, or stored query. A create statement in SQL creates an object inside of a relational database management system (RDBMS).

Syntax:

```
CREATE TABLE table_name
(
        Column_name1 data_type,
        Column_name2 data_type …
        Column_name_ndata_type
);
```

Example:

```
create table employee
(
empid number(10) primary key,
empname varchar2(20) not null,
dob date,
doj date,
dept char(10),
salary number(7)
);
```

Data Types :-

1. **String Data** - char, varchar, varchar2
2. **Numeric Data** - number, integer, float
3. **Temporal Data** – date, time, date & time combination
4. **Large Objects** - These are used for storing data objects like files and images:

   There are two types:

   - Character Large Objects (clobs)
   - Binary Large Objects (blobs)

Alter a Table:

To modify an existing database object. Alter the structure of the database.

a. To add a column in a table

Syntax: ALTER TABLE table_name ADD column_name datatype;

Example: SQL> alter table employee **add** exp number(2);

b. To delete a column in a table

Syntax: ALTER TABLE table_name DROP column column_nmae;

Example: SQL> alter table employee **drop column** exp;

c. To change the data type of a column in a table

Syntax: ALTER TABLE table_name MODIFY column_name datatype;

Example: SQL> alter table employee **modify** empname varchar2(30);

d. To rename a column in a table

Syntax: ALTER TABLE table_name RENAME COLUMN old_column TO new_column;

Example: SQL> alter table employee **rename column** empname to ename;

Table altered.

Rename Table:

Purpose: Rename an object

Syntax: RENAME old_table_name to new_table_name;

Example: rename employee to emp;

Describe Table:

Purpose: Describes the structure of the table

Syntax: DESC table_name;

Example: SQL> desc employee;

Name	Null?	Type
EMPID	NOT NULL	NUMBER(10)
ENAME	NOT NULL	VARCHAR2(20)
DOB		DATE
DOJ		DATE
DEPT		CHAR(10)
SALARY		NUMBER(7)

Drop Table:

Delete Objects from the Database

Syntax: DROP TABLE table_name;

Example: drop table employee;

Truncate Table:

Remove all records from a table, including all spaces allocated for the records are removed.

Syntax: TRUNCATE TABLE table_name;

Example: **truncate** table employee;

DATA MANIPULATION LANGUAGE (DML)

- Data manipulation language allows the users to query and manipulate data in existing schema in object.
- It allows following data to ***insert, delete, update and recovery*** data in schema object.

DML COMMANDS:

1. INSERT
2. UPDATE
3. DELETE
4. SELECT

QUERY:

- Query is a statement in the DML that request the ***retrieval of data from database***.
- The portion of the DML used in a Query is called Query language.
- The SELECT statement is used to query a database

INSERT Query:

- Values can be inserted into table using insert commands.
- There are two types of insert commands.

  (i). Multiple row inserts commands (using '&' symbol)

  (ii). Single row insert command (without using '&'symbol)

1) **To insert one row at a time.**

   Syntax: INSERT INTO table_name VALUES (value1, value2, value3,...);

   Ex: insert into employee values (10144, 'Kavitha', '12-APR-2000', '25-APR-2019', 'Develop', 61000);

2) **To insert many rows at a time**

   Syntax: INSERT INTO table_name values (&column1, &column2,….);

   Ex: insert into employee values (&empid, '&empname', '&dob', '&doj', '&dept', '&salary');

3) **Inserting Data's in specified columns:**

   Syntax: INSERT INTO table_name (col1, col2, ..,col-n) VALUES (val1, val2 ,…,val n);

   Example: insert into employee (empid, empname) values (10110, 'Sambasiva');

SELECT Query:

Retrieve (read) data from a database.

1. To select particular column(s) from a table.
2. To select all records from a table
3. To select particular record using WHERE clause.

1) **To select *particular column*(s) from a table.**

Syntax: SELECT column_name(s) FROM table_name;

Example: SQL> select empid, empname from employee;

```
     EMPID EMPNAME
---------- --------------------
     10159 Hima Shekar
     10166 Sriram
     10193 Harini
     10101 Harinath
     10144 Kavitha
     10240 Balakrishna
     10109 Krishna
     10167 Manideep
     10171 SivaSai
     10207 Bharathi
     10607 Harinath
     10236 Balachandar
     10244 Bhumesh
     10225 Sayadeevi
     10208 Arunkumar
     10216 Tharun T
     10235 Anilkumar
17 rows selected.
```

2) **To select *all records* from a table**

Syntax: SELECT * FROM table_name;

Example: SQL> select * from employee;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10159	Hima Shekar	12-NOV-99	30-MAR-15	Plan	65000
10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000

10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10171	SivaSai	30-NOV-98	05-APR-19	HR	72000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

17 rows

3) **To select *particular record* using WHERE clause**.

Syntax: SELECT * FROM table_name WHERE column = value;

Example: SQL> select * from employee where **empid=10167**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000

4) **Select using Distinct:**

- It is helpful when there is need of ***avoiding the duplicate values*** present in any specific columns/table.
- **Syntax:** SELECT DISTIN T column name(s) FROM table_name;
- **Example:** SQL> select **distinct dept** from employee;

DEPT
Develop
HR
Analyst
Plan
Sales

UPDATE Query:

- This command is used for updating or modifying the values of columns in a table (relation).

Syntax: UPDATE table_name SET column_name = new_value [WHERE condition];

Example: update employee set empid = '10159' where empname = 'hima sekar';

DELETE Query:

- This command is used for removing one or more records from a table (relation).

Syntax: DELETE FROM table_name [WHERE condition];

Example: delete from employee where empid = '10091';

DATA CONTROL LANGUAGE (DCL)

- DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
  - ✓ CREATE – Create user area in the database.
  - ✓ GRANT - Gives user's access privileges (Rights) to database.
  - ✓ REVOKE - Withdraw user's access privileges given by using the GRANT command.

Create User:

- **Syntax:** CREATE USER user_name IDENTIFIED BY password;
- **Example:** SQL> create user biher identified by cse123;

  User created.

Grant Command:

- **Syntax:** GRANT ALL PRIVILEGES TO User_name;
- **Example:** SQL> grant all privileges to biher;

  Grant succeeded.

  SQL> connect

  Enter user-name: biher

  Enter password:

  Connected.

Revoke Command:

- **Syntax:** REVOKE ALL PRIVILEGES FROM User_Name;
- **Example:** SQL> revoke all privileges from biher;

  Revoke succeeded.

  SQL> connect

Enter user-name: biher

Enter password:

ERROR:

ORA-01045: user BIHER lacks CREATE SESSION privilege; logon denied

Warning: You are no longer connected to ORACLE.

TRANSACTION CONTROL LANGUAGE (TCL): -

- Transaction Control Language (TCL) commands are used to ***manage transactions*** in the database.
- These are used to manage the changes made to the data in a table by DML statements.
- **TCL commands**:
  - ✓ **COMMIT**– Commits a Transaction.
  - ✓ **ROLLBACK**– Rollbacks a transaction in case of any error occurs.
  - ✓ **SAVEPOINT** – Sets a savepoint within a transaction.

Commit:

- COMMIT command is used to ***permanently save any transaction*** into the database.
- When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
- To avoid that, we use the COMMIT command to mark the changes as permanent.
- **Syntax:** COMMIT;

Savepoint:

- SAVEPOINT command is used to temporarily save a transaction so that you can roll back to that point whenever required.
- **Syntax:** SAVEPOINT Savepoint_Name;

Rollback:

- Restores the database to last committed state.
- It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.
- **Syntax:** ROLLBACK;

(or)

ROLLBACK TO Savepoint_Name;

CONSTRAINS:

It is a mechanism used to present invalid data entry into the table.

Types:

(i) Domain integrity constraints.

(ii) Entity integrity constraints

(iii) Referential integrity constraints

(i). Domain Integrity Constraints:

- Each table has certain ***set of columns*** and each column ***allows a same type of data***, based on its data type.
- The column ***does not accept values of any other data type***.

a) **NOT NULL constraint**:

- By default, a column can hold **NULL**.
- If you do not want to allow **NULL** value in a column, you can place the **NOT NULL** constraint on this column.
- The **NOT NULL** constraint specifies that **NULL** is not an allowable value.
- User has to provide a value for the column.
- **Example**:

```
CREATE TABLE student
(
        Regno       Number(10)    not null,
        Stuname     Varchar(25)   not null,
        Age         Number(3),
        Dept        Char(10)
);
```

b) **CHECK constraints**:

- The **CHECK** constraint ensures that ***all values in a column satisfy certain conditions***.
- Once defined, the database will allow the new value, if the new value satisfies the **CHECK** constraint.
- The **CHECK** constraint is used to ensure data quality.
- **Example**:

```
create table boys_hostel
(
        Hostel_id       Number(10)      not null,
        Regno           Number(10)      not null,
        Stuname         Varchar(25)     not null,
        Gender          Char(8)         CHECK(gender='male')
);
```

(ii). Entity Integrity Constraints:

- Entity integrity is concerned with ensuring that each row of a table has a unique and non-null value.
- There are two types of entity integrity constraints,
  1. **UNIQUE** - Unique constraints is used to prevent duplication of values and null values.
  2. **PRIMARY KEY** - This is used for to prevent duplication of values and null values.

a) UNIQUE Constraints:

Unique constraint is used to prevent duplication of values. It does not allow null value for the column.

Example:

```
create table boys_hostel
(       Hostel_id       Number(10)      unique,
        Regno           Number(10)      not null,
        Stuname         Varchar(25)     not null,
        Gender          Char(8)         check(gender='male')
);
```

b) PRIMARY KEY Constraint:

This is used for to prevent duplication of values .It will not allow null values for the column.

Example:

```
create table boys_hostel
(
        Hostel_id       Number(10)      primary key,
        Regno           Number(10)      not null,
        Stuname         Varchar(25)     not null,
        Gender          Char(8) check(gender='male')
);
```

(iii). Referential Integrity Constraint:

- To establish a ***parent child relationship*** between two tables having a common column we can use referential integrity constraints.
- To implement this, we should define the column in the ***parent table as primary key*** and the ***same column in the child table as a foreign key*** referring to the corresponding parent entry.

Example:

Parent Table:

```
create table student
(
        Regno           Number(10)      primary key,
        Stuname         Varchar(25)     not null,
        Dept            Char(10),
        Age             Number(3),
        Gender          Char
);
```

Child table:

```
create table boys_hostel
(       Hostel_id       Number(10)      primary key,
        Regno           Number(10)      references student (Regno),
        Stuname         Varchar(25)     not null,
        Gender          Char(8)         check(gender='male'));
```

(or**)**

```
create table boys_hostel
(
        Hostel_id       Number(10)    primary key,
        Vtuno           Number(10),
        Stuname         Varchar(25)   not null,
        Gender          Char(8) check(gender='male')
        foreign key(Regno) references student (Regno),
);
```

VIEWS:

- A View is a ***virtual table*** that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.
- If data is changing in the original table, the same change is reflected in the view.
- A view can be built on top of a ***single table or multiple tables***.
- It can also be ***built on top of another view***.

Advantages of Views:

1. **Ease of use**: A view hides the complexity of the database tables from end users.
2. **Space savings**: Views takes very little space to store, since they do not store actual data.
3. **Additional data security**: Views can include only certain columns in the table so that only the non-sensitive columns are included and exposed to the end-user.

a) Creating a View:

Syntax: CREATE VIEW view_name AS SELECT column_1, ….column_n FROM table_name;

Examples:

1. create view emp_view as select empid, empname, dept from employee;
2. create view emp_view as select * from employee;
3. create view emp_view as select e.empname, e.dept, *b.branch* from employee e, *branch b* where e.empid=b.empid;

b) Removing a View:

Syntax: DROP VIEW View_Name;

c) Updatable View:

For a view to be updatable, the DBMS must be able to trace any row or column back to its row or column in the source table.

Syntax: UPDATE view_table_name SET column="value" WHERE <CONDITION>;

Examples: SQL>update emp_view set empname='Kumara Swamy' where empid= 10190;

d) Select View:

```
select * from emp_view;
```

e) Insert Data into View:

```
SQL> insert into emp_view values (&empid, '&name', '& dept');
```

INDEX TABLE:

- I*ndexes* help us ***retrieve data*** from tables ***quicker***.
- ***Without an index***, the database system ***reads through the entire table*** to locate the desired information.
- ***With index***, the database system ***first go through the index table*** to find out where to retrieve the data, and then go to these locations ***directly to get the needed data***. *This is much faster*.

Syntax: CREATE INDEX index_name ON table_name (column_name(s));

Example:

```
create index emp_index on employee(empid);
create index emp_index on employee (empname, dob);
```

JOIN QUERIES:

- Join is a query in which data is returned from *two or more tables*.
- The purpose of a join is to combine the data spread across tables.
- **Syntax:** Select column_1, column_2, …..column_n from table1, table2 [where <condition>];
- **Types of Joins**:
  1. Natural Join
  2. Inner Join
  3. Left Outer Join
  4. Right Outer Join
  5. Full Join

Table Name: **Branch**

BNAME	SALES	PRODUCT	TXN_DATE
AVADI	35000	P1	20-JUL-2018
GUINDY	21000	P2	20-JUL-2018
AVADI	15000	P	21-JUL-2018
AMBATTURE	20000	P1	20-JUL-2018
T-NAGAR	32000	P1	21-JUL-2018
TAMBARAM	33000	P2	20-JUL-2018

Table Name**: Region**

Rname	Bname
WEST	AMBATTUR
WEST	T-NAGAR
EAST	AVADI
EAST	GUINDY
NORTH	ANNA NAGAR

1. Natural Join:

It returns the matching rows from the tables that are being joined.

Example: SQL> select b.bname from **branch b, region** where b.bname=region.bname;

BNAME

AVADI
GUINDY
AVADI
AMBATTUR
T-NAGAR

2. Inner Join:

- An inner join in SQL returns rows where there is *at least one match on both tables*.

Syntax: SQL> SELECT column_1, column_2,… column_n from table_1 INNER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.b_name, rname from branch b **inner join** region r on b.bname = r.bname;

BNAME	RNAME
AVADI	EAST
GUINDY	EAST
AVADI	EAST
AMBATTUR	WEST
T-NAGAR	WEST

3. Left Outer Join:

- In left outer join, all rows from the first table mentioned in the SQL query is selected, regardless whether there is a matching row on the second table mentioned in the SQL query.

Syntax: SQL> SELECT column_1, column_2, …., column_n FROM table_1 LEFT OUTER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.name, rname from branch b **left outer join** region r on b.bname=r.bname;

BNAME	REGNAME
AMBATTUR	WEST
T-NAGAR	WEST
AVADI	EAST
AVADI	EAST
GUINDY	EAST
TAMBARAM	

4. Right Outer Join:

- In right outer join, all rows from the second table mentioned in the SQL query is selected, regardless whether there is a matching row on the first table mentioned in the SQL query.

Syntax: SQL> SELECT column_1, column_2,…..column_n FROM table_1 RIGHT OUTER JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL>select b.bname, rname from branch b **right outer join** region r on b.bname=r.bname;

BNAME	RNAME
AVADI	EAST
GUINDY	EAST
AVADI	EAST
AMBATTUR	WEST
T-NAGAR	WEST
	NORTH

5. Full Join:

- It is the combination of both left outer and right outer join.

Syntax: SQL> SELECT column_1, column_2,…..column_n FROM table_1 FULL JOIN table_2 ON table_1.column=table_2.column;

Ex: SQL> select b.bname, rname from branch b **full join** region r on b.bname=r.bname;

BNAME	RNAME
AMBATTUR	WEST
T-NAGAR	WEST
AVADI	EAST
AVADI	EAST
GUINDY	EAST
TAMBARAM	
	NORTH

SUB QUERIES or NESTED SUB-QUERY:

- Queries, one within the other is termed as a **sub query**.
- **Sub-queries** are used to retrieve data from tables that depend on the values in the table itself.
- Sub-queries can reside in the **WHERE** clause, the **FROM** clause, or the **SELECT** clause.

Table Name: **EMPLOYEE**

EMPID	EMPNAME	BRNO	DEPTNO	SALARY	EXP
1101	GOUSE BASHA	111	1001	25000	5
1102	BALAJEE	111	1001	15000	1
1103	BHARATHI	222	4001	25000	5
1104	KAMATCHI	222	4001	20000	3
1105	MURTHI	333	5001	15000	1
1106	KANNAN	333	5001	20000	3
1107	MALADHI	444	5001	20000	3
1108	ARUN KUMAR	444	3001	15000	1
1109	EDWARD	111	2001	20000	4
1110	SUMANTH	333	2001	25000	6

DEPARTMENT

DNO	DNAME
1001	IT
2001	ACCOUNTS
3001	SALES
4001	SERVICE
5001	PLANNING

BRANCH

BNO	BNAME
111	CHENNAI
222	BANGALURE
333	HYDERABAD
444	MUMBAI

(a). Insert with Sub-Queries:

- INSERT statement can be used with subqueries.

Syntax: INSERT INTO table_name [(column1 [, column2])] SELECT *|column1 [, column2] FROM table1 [, table2] [WHERE VALUE OPERATOR];

Ex: SQL>insert intochennai_branch select * from employee where brno = (select bno from branch where bname='chennai');

```
SQL> select * from chennai_branch;

     EMPID EMPNAME                        BRNO      DEPTNO      SALARY         EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1101 GOUSE BASHA                    111        301       25000          5
      1102 BALAJEE                        111        301       15000          1
```

(b). Update with Sub-Queries:

- In an UPDATE statement, we can set new column value equal to the result returned by a single row subquery.

Syntax: UPDATE table_name SET column_name = new_value WHERE OPERATOR (SELECT column_name FROM table_name) [WHERE <Condition>];

Example: SQL>UPDATE employee SET salary = salary+2000 WHERE deptno = (SELECT dno FROM department WHERE dname='sales');

EMPID	EMPNAME	BRNO	DEPTNO	SALARY	EXP
1101	GOUSE BASHA	111	1001	25000	5
1102	BALAJEE	111	1001	15000	1
1103	BHARATHI	222	4001	25000	5
1104	KAMATCHI	222	4001	20000	3
1105	MURTHI	333	5001	15000	1
1106	KANNAN	333	5001	20000	3
1107	MALADHI	444	5001	25000	5
1108	ARUN KUMAR	444	3001	17000	1
1109	EDWARD	111	2001	20000	4
1110	SUNANTHA	333	2001	25000	6

(c). <u>Delete with Sub-Queries</u>

- DELETE statement can be used with subqueries.

Syntax: DELETE FROM table_name WHERE OPERATOR (SELECT Column_name FROM table_name) [WHERE) <condition>];

Example: SQL>delete from employee where deptno=(select dno from department where dname=’it’);

```
SQL> SELECT * FROM EMPLOYEE;

     EMPID EMPNAME                    BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

(d). <u>SQL IN with Sub-Queries</u>

- The IN operator allows you to specify multiple values in a WHERE clause.

Syntax*:* SELECT column_name(s) FROM table_name WHERE column_name IN (SELECT Column_name FROM table_name WHERE condition);

Example: SQL> select empname, bname from employee join branch on brno=bno and brno in (select bno from branch where bnamein('chennai','mumbai‘));

```
EMPNAME              BRNO BNAME                           BNO
-------------------- ---------- ------------------------------ ----------
MALADHI                 444 MUMBAI                        444
ARUN KUMAR              444 MUMBAI                        444
EDWARD                  111 CHENNAI                       111
```

(e). Sql NOT IN with Sub-Queries:

- NOT IN ( ) makes sure that the expression proceeded does not have any of the values present in the arguments

<u>Syntax</u>: SELECT column_name(s) FROM table_name WHERE column_name NOT IN (SELECT Column_name FROM table_name WHERE condition);

Example: SQL> SELECT empname, brno, bname, bno FROMemployee JOIN branch ON brno=bno AND brno NOT IN (SELECT bno FROM branch WHERE bname IN ('chennai', 'mumbai'));

EMPNAME	BRNO	BNAME	BNO
BHARATHI	222	BANGALURE	222
KAMATCHI	222	BANGALURE	222
MURTHI	333	HYDERABAD	333
KANNAN	333	HYDERABAD	333
SUNANTHA	333	HYDERABAD	333

(f). Sql EXIST with Sub-Queries:

- The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns true if the subquery returns one or more records.

Syntax: SELECT column_name(s) FROM table_name WHERE EXISTS (SELECT column_name FROM table_name WHERE condition);

Example:

```
SQL> SELECT * FROM EMPLOYEE WHERE EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SPARES');

no rows selected
```

```
SQL> SELECT * FROM EMPLOYEE WHERE EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SALES');

     EMPID EMPNAME                BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

(g). NOT EXIST with Sub-Queries:

- The NOT condition can be combined with the EXISTS condition to create a NOT EXISTS condition.

Syntax: SELECT column_name(s) FROM table_name WHERE NOT EXISTS (SELECT Column_name FROM table_name WHERE condition);

Example:

```
SQL> SELECT * FROM EMPLOYEE WHERE NOT EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='SALES');

no rows selected
```

```
SQL> SELECT * FROM EMPLOYEE WHERE NOT EXISTS(SELECT DNO FROM DEPARTMENT WHERE DNAME='TAX');

     EMPID EMPNAME                BRNO     DEPTNO     SALARY        EXP
---------- -------------------- ---------- ---------- ---------- ----------
      1103 BHARATHI                    222       4001      25000          5
      1104 KAMATCHI                    222       4001      20000          3
      1105 MURTHI                      333       5001      15000          1
      1106 KANNAN                      333       5001      20000          3
      1107 MALADHI                     444       5001      25000          5
      1108 ARUN KUMAR                  444       3001      17000          1
      1109 EDWARD                      111       2001      20000          4
      1110 SUNANTHA                    333       2001      25000          6

8 rows selected.
```

AGGREGATE FUNCTIONS:

i) Count(): To count the no of records, based on given attribute.

Syntax: SELECT COUNT(column_name) FROM table_name;

ii) max(): To identify the maximum value

Syntax: SELECT MAX(column_name) FROM table_name;

iii) min(): To identify the minimum value

Syntax: SELECT MIN(column_name) FROM table_name;

iv) sum(): To calculate summation of a particular column

Syntax: SELECT SUM(column_name) FROM table_name;

v) avg(): To calculate average of a particular column

Syntax: SELECT AVG(column_name) FROM table_name;

SET OPERATIONS

1. Union:

Purpose: Used to combine all values present in two relations but the duplicates are removed

Command: UNION

Syntax:

SELECT column_name1,….column_name_n FROM table_name_1 UNION select column_name1….column_name_n FROM table_name_2;

2. Union All:

Purpose: Used to combine all values present in two relations.

Command: UNION

Syntax:

SELECT column_name1,….column_name_nFROM table_name_1 UNION ALL SELECT column_name1….column_name_n FROM table_name_2;

3. Intersect:

Purpose: Combine the common values in two relation

Command : INTERSECT

Syntax:

SELECT column_name1,….column_name_n FROM table1_name_1
INTERSECT SELECTcolumn_name1….column_name_n FROM table_name_2;

4. Minus

Purpose: Includes the tuples from first_table but that should not in second_table

Command: MINUS

Syntax: SELECT column_name1,….column_name_n FROM table1_name_1
MINUS SELECTcolumn_name1….column_name_n FROM table_name_2;

NORMALIZATION:

- Normalization is the ***process of organizing the data*** in the database.
- Normalization is used to ***minimize the redundancy*** from a relation or set of relations. It is also used to ***eliminate*** the undesirable characteristics like ***Insertion, Update and Deletion Anomalies***.
- Normalization divides the ***larger table into the smaller*** table and links them using relationship.
- The ***normal form*** is used to reduce redundancy from the database table.

ANOMALIES IN DBMS

- There are ***three types of anomalies*** that occur when the database is not normalized.
- These are – Insertion, update and deletion anomaly.
- **Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: ***emp_id*** for storing employee's id, ***emp_name*** for storing employee's name, ***emp_address*** for storing employee's address and ***emp_dept*** for storing the department details in which the employee works. At some point of time the table looks like this:

EMP_ID	EMP_NME	EMP_ADDRESS	EMP_DEPT
10196	RAKESH	CHENNAI	D0011
10196	RAKESH	CHENNAI	D002
10198	NITYANAND	HYDERABAD	D890
10203	PAVAN KUMAR	GUNTUR	D900
10203	PAVAN KUMAR	GUNTUR	D004

- The above table is ***not normalized***.

1) **Update anomaly**:
   - ❖ In the above employee table, we have ***two rows for employee Rakesh*** as he belongs to two departments of the company.
   - ❖ If we ***want to update the address of Rakesh*** then we ***have to update the same in two rows*** or the data will become inconsistent.

2) **Insert anomaly**:
   - ❖ Suppose a ***new employee joins the company***, who is under training and currently ***not assigned to any department*** then we would ***not be able to insert the data*** into the table if emp_dept field doesn't allow nulls.

3) **Delete anomaly**:

- ❖ Suppose, if at a point of time the ***company closes the department D890*** then deleting the rows that are having emp_dept as D890 would ***also delete the information of employee Nityanand*** since she is assigned only to this department.
- ❖ To overcome these anomalies, we need to normalize the data.

Functional Dependency in DBMS:

- "The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table".
- **For example:** Suppose we have a ***student table*** with attributes: ***Stu_Id***, ***Stu_Name***, ***Stu_Age***. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as **functional dependency** and can be written as ***Stu_Id->Stu_Name*** or in words we can say ***Stu_Name is functionally dependent on Stu_Id.***

Types of Functional Dependencies

1. Trivial functional dependency
2. non-trivial functional dependency
3. Multivalued dependency
4. Transitive dependency

1) Trivial Functional Dependency:

- The dependency of an attribute on a set of attributes is known as ***trivial functional dependency*** if the set of attributes includes that attribute.

Example: Consider a table with two columns **Student_Id** and **Student_Name**.

- **{Student_Id, Student_Name} -> Student_Id** is a trivial functional dependency as **Student_Id is a subset of {Student_Id, Student_Name}**.
- Also, **Student_Id -> Student_Id** & **Student_Name -> Student_Name** are trivial dependencies too.

2) Non-Trivial Functional Dependency:

- If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:

- An employee table with three attributes: **emp_id, emp_name,** and **emp_address**.
- The following functional dependencies ***are non-trivial***:
  - ✓ emp_id -> emp_name (emp_name is not a subset of emp_id)
  - ✓ emp_id -> emp_address (emp_address is not a subset of emp_id)
- **On the other hand, the following dependencies are trivial:**
  - ✓ {emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
- **Completely non trivial FD**:
  - ✓ If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

3) Multivalued dependency:

- Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.
- **For example**: Consider a bike manufacture company, which produces two colors (Black and Red) in each model every year.
- Here columns **manuf_year and color are independent** of each other and **dependent on bike_model**.
- In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:
  - ✓ **bike_model ->> manuf_year**
  - ✓ **bike_model ->> color**

BIKE_MODEL	MANUF_YEAR	COLOR
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

4) Transitive Dependency:

- A functional dependency is said to be transitive if it is ***indirectly formed by two functional dependencies***. For e.g.
- X -> Z is a transitive dependency if the following three functional dependencies hold true:
  - ✓ X->Y
  - ✓ Y does not ->X
  - ✓ Y->Z
- **Note:** A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Example:

BOOK	AUTHOR	AUT_AGE
Game of Thrones	George R.R Martin	66
Harry Potter	J. K. Rowling	49
Dying of the Light	George R.R Martin	66

- ✓ {Book} ->{Author} (if we know the book, we know the author name)
- ✓ {Author} does not ->{Book}
- ✓ {Author} -> {Aut_age}

- Therefore, as per the rule of **transitive dependency**: {Book} -> {Aut_age} should hold, that makes sense because if we know the book name we can know the author's age.

Normalization Types:

Here are the most commonly used normal forms:

1) First normal form(1NF)
2) Second normal form(2NF)
3) Third normal form(3NF)
4) Boyce & Codd normal form (BCNF)

First Normal Form (1NF)

- As per the rule of first normal form, an attribute (column) of a table ***cannot hold multiple values.*** It should hold ***only atomic values***.
- **Example:** Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_MOBILE
10211	Ramesh	Delhi	8912312390
			8812121212
10098	Bharani	Kanpur	9900012222
10127	Abhiram	Chennai	7778881212
			9990000123
10093	Harini	Bangalore	8123450987

- Two employees (**Ramesh & Abhiram) are having two mobile numbers** so the company stored them in the same field as you can see in the table above.
- This **table is not in 1NF** as the rule says "***each attribute of a table must have atomic*** (single) ***values***", the emp_mobile values for employees **Ramesh & Abhiram** violates that rule.
- To make the table complies with 1NF we should have the data like this:

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_MOBILE
10211	Ramesh	Delhi	8912312390
10211	Ramesh	Delhi	8812121212
10098	Bharani	Kanpur	9900012222
10127	Abhiram	Chennai	7778881212

10127	Abhiram	Chennai	9990000123
10093	Harini	Bangalore	8123450987

Second Normal Form (2NF)

- A table is said to be in 2NF if both the following conditions hold:
  1) Table is in 1NF (First normal form)
  2) No non_key attribute is dependent on the any candidate key of table.
- **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.

Faculty_ID	Subject	Faculty_Age
111	DBMS	38
111	CMC	38
222	PCD	34
222	TOC	34
333	OS	36
333	DSC	36

Candidate Keys: {Faculty_ID, Subject}

Non_key attribute: Faculty_Age

- The ***table is in 1 NF*** because each attribute has atomic values.
- However, ***it is not in 2NF*** because ***non_key attribute Faculty_Age is dependent on Faculty_ID alone*** which is a proper subset of candidate key.
- This violates the rule for 2NF.
- To make the table complies with 2NF ***we can break it in two tables*** like this: **Faculty_Details** table and **Faculty_Subject** table**:**

Faculty_ID	Faculty_Age
111	38
222	34

333	36

Faculty_ID	Subject
111	DBMS
111	CMC
222	PCD
222	TOC
333	OS
333	DSC

Note: Now the tables are in Second normal form (2NF).

Third Normal Form (3NF)

- A table design is said to be in 3NF if both the following conditions hold:
  1. Table must be in 2NF
  2. Transitive functional dependency of non-prime attribute on any super key should be removed.
- An attribute that is not part of any candidate key is known as non-prime attribute.
- In other words, 3NF can be explained like this:

Example: Suppose a company wants to store the complete address of each employee, they create a table named **employee_details** that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
10118	Tharuni	504001	TS	Adilabad	Adilabad
10128	Vinay	600045	TN	Thambaram	Chennai
10146	Kiran Sai	533001	AP	Kakinda	East Godhavari
10248	Vishnuvardhan	523001	AP	Ongole	Prakasham
10235	Anil Kumar	521001	AP	Machilipatnam	Krishna

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip} and so on.
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:

employee table:

emp_id	emp_name	emp_zip
10118	Tharuni	504001
10128	Vinay	600045
10146	Kiran Sai	533001
10248	Vishnuvardhan	523001
10235	Anil Kumar	521001

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
504001	TS	Adilabad	Adilabad
600045	TN	Thambaram	Chennai
533001	AP	Kakinda	East Godhavari
523001	AP	Ongole	Prakasham
521001	AP	Machilipatnam	Krishna

Boyce Codd normal form (BCNF)

- 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:

emp_id	emp_city	emp_dept	dept_no	dept_no_of_emp
1001	Chennai	Production and planning	D001	200
1001	Chennai	stores	D001	250
1002	Hyderabad	design and technical support	D134	100
1002	Hyderabad	Purchasing department	D134	600

- **Functional dependencies in the table above:**
  - ✓ emp_id -> emp_nationality
  - ✓ emp_dept -> {dept_no, 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:

emp_city:

emp_id	emp_city
1001	Chennai
1002	Hyderabad

emp_dept:

emp_dept	dept_no	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

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.

***********BEYOND THE SYLLABUS********

SQL – WHERE Clause: -

- WHERE keyword is used for ***fetching (Read) filtered*** data in a result set.
- It is used to fetch data according to a ***particular criterion***.
- **Syntax:**

  SELECT column1, column2 FROM table_name WHERE column_name OPERATOR value;

List of operators that can be used with where clause:

OPERATOR	DESCRIPTION
>	Greater Than
>=	Greater than or Equal to
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Ex: 1

SQL> select * from employee where **empid=10167**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000

Ex: 2

SQL> select * from employee where **dob>='01-Jan-2000'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000

10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

10 rows selected.

WHERE clause with - *AND* Operator: -

- The WHERE clause when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.
- **Syntax:** SELECT * FROM table_name WHERE condition1 AND condition2;
- **Example:** SQL> select * from employee where **empname = 'Harinath' and dept = 'Sales'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600

WHERE Clause - BETWEEN Operator:-

- BETWEEN can be used to get those items that fall within a range.
- **Syntax:** SELECT column1 [,columns] FROM table_name WHERE column_name BETWEEN value1 AND value2;
- **Ex:** SQL> select * from employee where **salary between 50000 and 60000**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000

7 rows selected.

WHERE Clause - LIKE Operator: -

- LIKE clause compares data with an expression using *wildcard operators* to match pattern given in the condition.
- Wildcard Operators: There are 2 wildcard operators that are used in LIKE clause.
  - ✓ Percent sign" **%**": Represents zero, one or more than one character.

✓ Underscore sign" _": Represents only a single character.

Example:

SQL> select * from employee where **empname like '_a%'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000

8 rows selected.

SQL> select * from employee where **salary like '_7%'**;

EMPID	EMPNAME	DOB	DOJ	DEPT	SALARY
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000

SQL – ORDER BY Clause: -

- Order by clause is used with SELECT statement for **arranging retrieved data in sorted order**.
- The Order by clause **by default sorts the retrieved data in ascending order**. To sort the data in descending order DESC keyword is used with Order by clause.
- **Syntax:** SELECT column-list|* FROM table-name ORDER BY ASC | DESC;
- **Example:** SQL> select * from employee **order by empid**;

EMPID	**EMPNAME**	**DOB**	**DOJ**	**DEPT**	**SALARY**
10101	Harinath	06-NOV-00	31-MAR-17	Develop	57000
10109	Krishna	28-MAY-00	12-APR-16	Sales	60000
10144	Kavitha	12-APR-00	25-APR-19	Develop	61000
10159	Hima Shekar	12-NOV-99	30-MAR-15	Plan	65000

10166	Sriram	23-FEB-99	04-APR-18	Develop	60000
10167	Manideep	12-APR-98	23-MAR-19	Develop	65000
10171	SivaSai	30-NOV-98	05-APR-19	HR	72000
10193	Harini	10-JUL-00	31-DEC-16	HR	55000
10207	Bharathi	20-MAR-00	31-JAN-18	HR	67000
10208	Arunkumar	03-JAN-99	20-SEP-18	Develop	67000
10216	Tharun T	28-MAY-01	14-JAN-15	Develop	65000
10225	Sayadeevi	27-JUL-00	15-JAN-19	HR	65000
10235	Anilkumar	22-JUL-01	18-DEC-18	Plan	56000
10236	Balachandar	29-JUN-98	31-DEC-19	Analyst	56000
10240	Balakrishna	08-JUN-01	16-MAY-17	Analyst	65000
10244	Bhumesh	20-MAR-98	30-NOV-17	Develop	60000
10607	Harinath	28-MAY-00	31-DEC-17	Sales	47600

17 rows selected.

SQL – GROUP BY Clause: -

- Group by clause is used to group the results of a SELECT query based on one or more columns.
- It is also used with SQL functions to group the result from one or more tables.
- group by is used to group different row of data together based on any one column.

Example:

SQL> select dept, sum(salary) from employee **group by dept**;

DEPT	**SUM(SALARY)**
----------	----------
Develop	435000
HR	259000
Analyst	121000
Plan	121000
Sales	107600

SQL – HAVING Clause: -

- Having clause is used with SQL Queries to **give more precise condition** for a statement.
- It is used to mention a **condition in Group by based SQL queries**, just like WHERE clause is used with SELECT query.

- **Syntax:** SELECT column_name, function(column_name) FROM table_name GROUP BY column_name HAVING condition;
- **Example:** SQL> select dept, sum(salary) from employee group by dept having sum(salary)<200000;

DEPT	SUM(SALARY)
Analyst	121000
Plan	121000
Sales	107600

SELECT INTO statement:

The SELECT INTO statement is most often used to create backup copies of tables or for archiving records.

Syntax: SELECT Column_name(s) INTO variable_name(s) FROM table_name WHERE condition;

❖ **To Select NULL values**:

We can use the SELECT statement to select the 'null' values also. For retrieving roes where some of the columns have been defined as NULLs there is a special comparison operator of the form IS[NOT]NULL.

Syntax: SELECT column name FROM table_name WHERE Column name IS NULL;

TRIGGERS:

- ❖ A **trigger** is a special type of stored procedure that automatically executes when an event occurs.
- ❖ Triggers are, written to be executed in response to any of the following events −
- ❖ A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- ❖ A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- ❖ A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- ❖ Triggers can be defined on the table, view, schema, or database with which the event is associated.

BENEFITS OF TRIGGERS:

- To generate data automatically.
- To enforce complex integrity constraints. (E.g. checking with sysdate, checking with data in another table).
- To customize complex security authorizations.
- To maintain replicate tables.
- To audit data modifications.

CREATING TRIGGERS:

Syntax:

CREATE OR REPLACE TRIGGER <**trigger_name**>
[BEFORE/AFTER] [INSERT/UPDATE/DELETE] ON <**table_name**>
[FOR EACH STATEMENT/FOR EACH ROW]
[WHEN <**condition**>]
PL/SQL BLOCK;

PARTS OF A TRIGGER

A database trigger has three parts, namely, a trigger statement, a trigger body and a trigger restriction.

a) **Trigger Statement:**
   A trigger statement specifies the DML statements like update, delete and insert and it fires the trigger body. It also specifies the table to which the trigger is associated.
b) **Trigger Body:**
   Trigger body is a PL/SQL block that is executed when a triggering statement is issued.
c) **Trigger Restriction:**
   Restrictions on a triggers can be achieved using the WHEN clause as shown in the syntax for creating triggers. They can be included in the definition of a row trigger, where in, the condition in the WHEN clause is evaluated for each row that is affected by the trigger.

TYPES OF TRIGGER:

Triggers are categorized into the following types based on when they are fired:

- Before
- After
- For each row
- For each statement (default)

Examples:

Q1: Write a PL/SQL program to create a trigger before the user inserts the data into the table.

```
SQL>CREATE or REPLACE TRIGGER INS1
        BEFORE INSERT ON employee
            BEGIN
                RAISE_APPLICATION_ERROR (-20001,'you can't insert a row');
            END;
```

OUTPUT:

```
SQL>insert into employee values(&emp_id, '&emp_name', '&dob', '&addr', '&sex', '&desig',
&deptno, '&maritsta', &salary);
ERROR at line 1:
ORA-20001: you cant insert a row
ORA-06512: at "CSE382.ins1", line 2
ORA-04088: error during execution of trigger 'CSE382.INS1'
```

Q2: Write a PL/SQL program to create a trigger before the user deletes the data from the table.

```
SQL>CREATE or REPLACE TRIGGER DEL1
        BEFORE INSERT ON employee
            BEGIN
                RAISE_APPLICATION_ERROR (-20001,'you can't  delete');
            END;
```

OUTPUT:

```
SQL>delete from employee where emp_id=4444;
delete from employee where emp_id=4444;
```

*

ORA-20001: **you can't delete**

ORA-06512: at "CSE382.DEL1", line 2

ORA-04088: error during execution of trigger '**CSE382.DEL1**'

Q3: Write a PL/SQL program to create a trigger before the user changes the value of the salary of the employee.

```
CREATE TRIGGER UPD1 BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
        IF :new.sal< 1000 THEN
                        RAISE_APPLICATION_ERROR(-20001,'SALARY CAN'T BE LOW
THAN THIS');
        END IF;
 END;
  /
TRIGGER CREATED.
```

OUTPUT:

SQL> UPDATE **employee** SET **sal=500** WHERE **dno=2**;

UPDATE **employee** SET **sal=500** WHERE **dno=2**

*

ERROR at line 1:

ORA-20001: **salary can't be low than this**

ORA-06512: at "**CSE382.UPD1**", line 3

ORA-04088: error during execution of trigger '**CSE382.UPD1**'

EMBEDDED SQL

1. SQL provides a powerful declarative query language. However, access to a database from a general-purpose programming language is required because,

   - SQL is not as powerful as a general-purpose programming language. There are queries that cannot be expressed in SQL, but can be programmed in C, FORTRAN, Pascal, COBOL, etc.
   - Non-declarative actions -- such as printing a report, interacting with a user, or sending the result to a GUI -- cannot be done from within SQL.

1. Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language (host language).
2. The mixture of SQL and general purpose programming languages is called embedded SQL.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<sqlca.h>
void main()
{
    EXEC SQL
    EXEC SQL BEGIN DECLARE SECTION;
intOrderID;        /* Employee ID (from user)        */
intCustID;          /* Retrieved customer ID        */
charSalesPerson[10];   /* Retrieved salesperson name      */
```

```
char Status[6];       /* Retrieved order status       */
    EXEC SQL END DECLARE SECTION;
    /* Prompt the user for order number */
    printf("Enter order number: ");
    scanf("%d", &OrderID);
    /* Execute the SQL query */
    EXEC SQL SELECT CustID, SalesPerson, Status FROM Orders
    WHERE OrderID= :OrderID
INTO :CustID, :SalesPerson, :Status;

    /* Display the results */
    printf("Customer number:%d\n",:CustID);
    printf ("Salesperson: %s\n",:SalesPerson);
    printf ("Status: %s\n", :Status);
    exit();
}
```

UNIT-III DATA STORAGE AND QUERYING 9

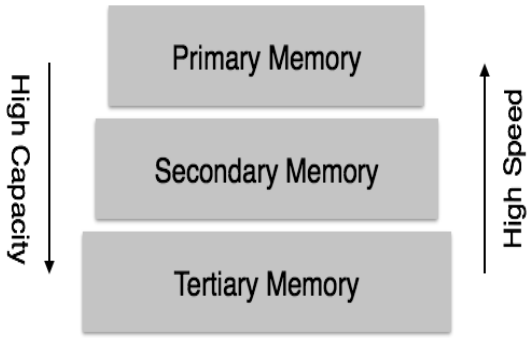
Storage and File Structure - File Organization - Indexing and Hashing- Ordered Indices- Static Hashing- Dynamic Hashing- Query Processing- Overview- Measures of Query Cost-Selection- Sorting- Join Operation- Evaluation of Expressions- Query Optimization- Overview- Transformation of Relational Expressions- Estimating Statistics of Expression Results.

STORAGE SYSTEM:

» Databases are stored in file formats, which contain records.

» At physical level, the actual data is stored in electromagnetic format on some device.

» These storage devices can be broadly categorized into three types –



- **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
- **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
- **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

Memory Hierarchy:

- A computer system has a well-defined hierarchy of memory. A CPU has direct access to it main memory as well as its inbuilt registers.
- The access time of the main memory is obviously less than the CPU speed.
- To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.
- The memory with the fastest access is the costliest one.
- Larger storage devices offer slow speed and they are less expensive; however, they can store huge volumes of data as compared to CPU registers or cache memory.

Magnetic Disks:

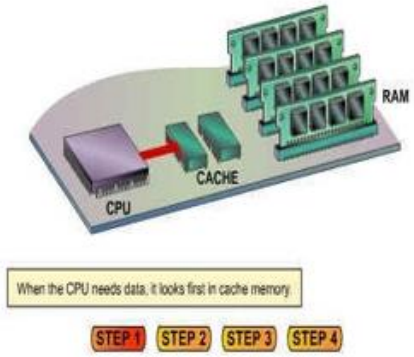
- Hard disk drives are the most common secondary storage devices in present computer systems.
- These are called magnetic disks because they use the concept of magnetization to store information.
- Hard disks consist of metal disks coated with magnetizable material. These disks are placed vertically on a spindle.
- A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one).
- Hard disks are formatted in a well-defined order to store data efficiently.
- A hard disk plate has many concentric circles on it, called **tracks**. Every track is further divided into **sectors**. A sector on a hard disk typically stores 512 bytes of data.

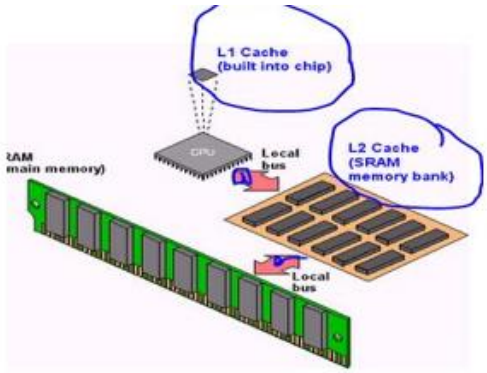
Several types of data storage exist in most computer systems.

They vary in ***speed of access***, ***cost per unit of data***, and ***reliability***. Following are the typical available storage media:

1. Cache Memory.
2. Main Memory.
3. Flash Memory.
4. Magnetic-Disk Storage.
5. Optical Storage.
6. Tape Storage.

1. **Cache Memory:**
   - ❖ Most costly and fastest form of storage.
   - ❖ Usually very small, and managed by the operating system.

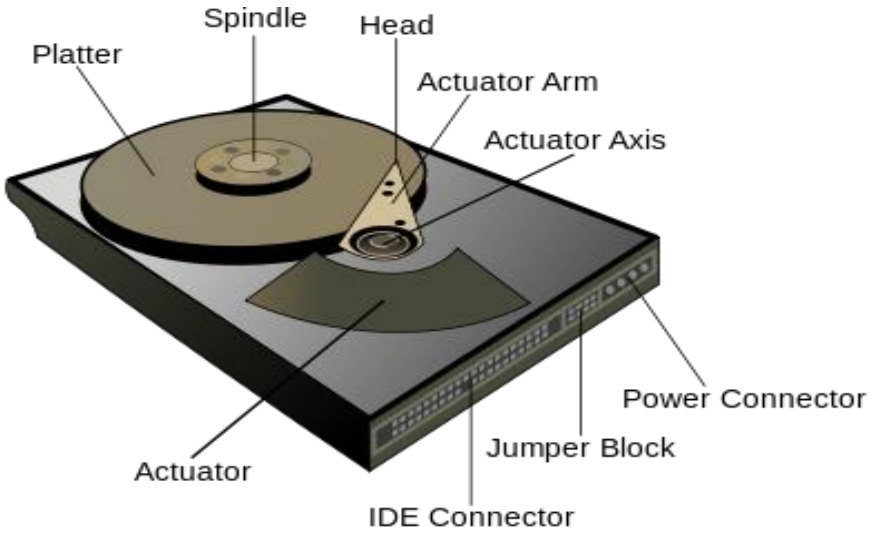




2. **Main Memory:**
   - ❖ **Primary storage**, also known as **main memory.**
   - ❖ Main Memory **for quick access** by the computer's processor.
   - ❖ Copy data from **Storage device into main memory**.
   - ❖ Main memory is closely connected to the processor, so moving instructions and data into and out of the **processor is very fast**.
   - ❖ The contents of Main Memory are **usually lost** if a power failure or system crash occurs.

3. **Flash Memory:**
   - ❖ Flash memory is a **non-volatile memory chip**. It is also known as Erasable Programmable Read-Only Memory (**EEPROM**)
   - ❖ Transferring data between a personal computer (PC) and digital devices.
   - ❖ It has the ability to be electronically **reprogrammed and erased**.
   - ❖ It is often found in USB **flash drives**, **MP3 players**, **digital cameras** and **solid-state drives**.
   - ❖ This technology also is useful for computer basic input/output systems (**BIOS**), (**PCMCIA**) cards, modems and video **game cards**.

4. **Magnetic-Disk Storage:**
   - ❖ The primary medium for the **long-term on-line storage** of data is the magnetic disk.
   - ❖ Usually, the **entire database** is stored on magnetic disk (HDD).
   - ❖ The system must move the data from **disk to main memory** so that they can be accessed.
   - ❖ The size of the magnetic disks currently ranges from a few **GB to 512 TB** (Available in Market).
   - ❖ Disk Storage **survives power failures** and system crashes.



5. **Optical Storage Device:**
   - ❖ Optical storage, electronic storage medium that uses **low-power laser beams** to record and retrieve digital (binary) data.

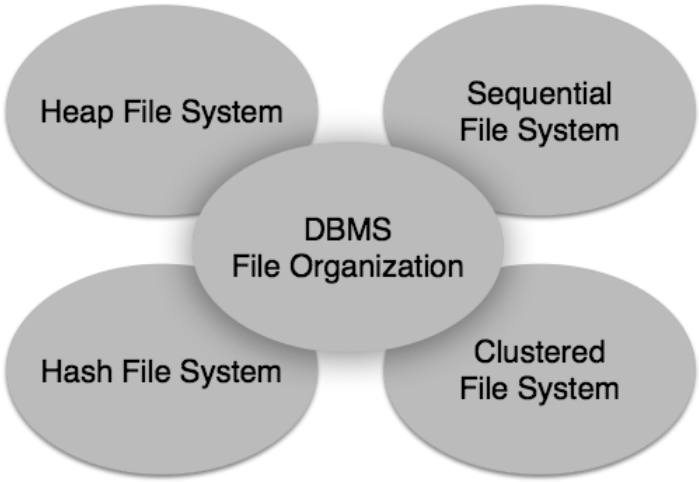
- The most popular forms of optical storage are **Compact Disk** (CD -750MB), **Digital Video Disk** (DVD 4.7 or 8.5 GB).
- **Write-once** versions of CD and DVD are called CD-R and DVD-R respectively.
- **Multiple-write** versions of CD and DVD are called CD-RW and DVD-RW respectively.

6. Tape Storage Device:

- Tape storage is used primarily for **Backup** and **Archival data**.
- **Cheaper**, but much **slower access**, since tape must be read **sequentially** from the beginning.
- Tape storage is referred as sequential-access storage
- Used as protection from disk failures.

FILE ORGANIZATION:

» File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records –



1) Heap File Organization:

» When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details.

» File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records.

» Heap File does not support any ordering, sequencing, or indexing on its own.

2) Sequential File Organization

» Every file record contains a data field (attribute) to uniquely identify that record.

» In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

» Practically, it is not possible to store all the records sequentially in physical form.

3) Hash File Organization

» Hash File Organization uses Hash function computation on some fields of the records.

» The output of the hash function determines the location of disk block where the records are to be placed.

4) Clustered File Organization

- Clustered file organization is not considered good for large databases.
- In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations:

Operations on database files can be broadly classified into two categories −

- ✓ **Update Operations**
- ✓ **Retrieval Operations**

- **Update operations** change the data values by insertion, deletion, or update.
- **Retrieval operations**, on the other hand, do not alter the data but retrieve them after optional conditional filtering.
- In both types of operations, **selection** plays a significant role.
- Other than creation and deletion of a file, there could be **several operations**, which can be done on files.

1) **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
2) **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
3) **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
4) **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
5) **Close** − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
    1) removes all the locks (if in shared mode),

2) saves the data (if altered) to the secondary storage media, and
3) releases all the buffers and file handlers associated with the file.

» The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

RAID (Redundant Array of Independent Disk)

❖ RAID is a way of storing the **same data in different places on multiple hard disks** to protect data in the case of a drive failure.

❖ RAID has an arrangement of several independent disks that are organized to improve ***reliability*** and at the same time increase ***performance***.

1) **Improved Performance:**
   - ✓ The Data is segmented into **equal-size partitions** which are transparently distributed across **multiple disks.**
   - ✓ The **Data Stripping** improves overall **I/O performance** and also **balances the load** among disks.
2) **Improved Reliability:**
   - ✓ Storing **redundant information** across the disks using a *parity scheme* or an *error-correcting scheme* to **improve the reliability**.
   - ✓ ***Parity Scheme*****:** - Each byte may have a parity bit associated with that record.

❖ RAID consists of an array of disks in which multiple disks are connected together to achieve different goals.

❖ RAID levels define the use of disk arrays.

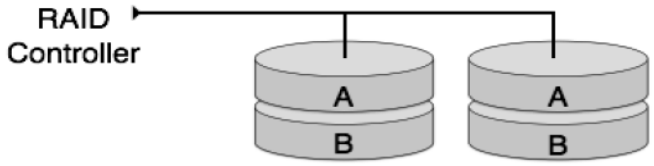
1. **RAID-0 (Non-redundant)**
2. **RAID-1 (Mirrored)**
3. **RAID-2 (Memory-Style Error-Correcting Code)**
4. **RAID-3 (Bit Interleaved Parity)**
5. **RAID-4 (Block Interleaved Parity)**
6. **RAID-5 (Bit Interleaved – Distributed Parity)**
7. **RAID-6 (P+Q Redundancy)**

RAID 0 (Non-redundant):

- A **striped array of disks** is implemented.
- The **data is broken** into blocks and the blocks are **distributed among disks**.
- Each disk receives a block of data to **write/read in parallel**.
- It enhances the **speed** and **performanc**e of the storage device.
- There is **no parity** and **no backup** in Level 0.
- It offers the best performance, but **no-fault tolerance**.

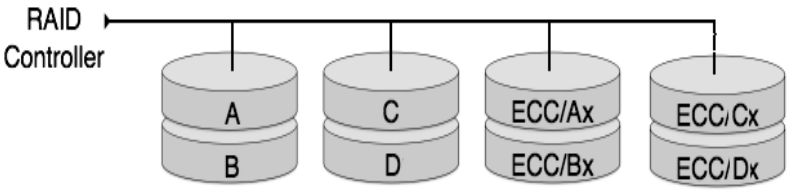
RAID 1 (Mirrored):

- RAID 1 uses **mirroring techniques**. When data is sent to a RAID controller, it sends a **copy of data to all the disks** in the array.
- RAID -1 provides **100% redundancy** in case of a failure.
  - ✓ **Read performance is improved** since either disk can be read at the same time.
  - ✓ **Write performance is the same** as for single disk storage.



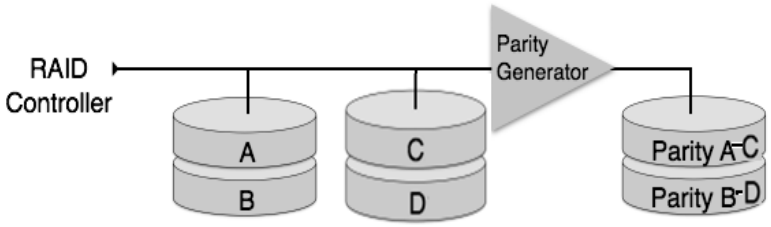
RAID 2 (Memory-Style Error-Correcting Code):

- RAID 2 generate **Error Correction Code** using Hamming distance for its data, **striped on different disks**.
- The **data is broken** into blocks and the blocks are **distributed among disks** and **ECC codes** of the data are stored on **different set disks**.
- Due to its **complex structure** and **high cost**, RAID 2 is no longer used.



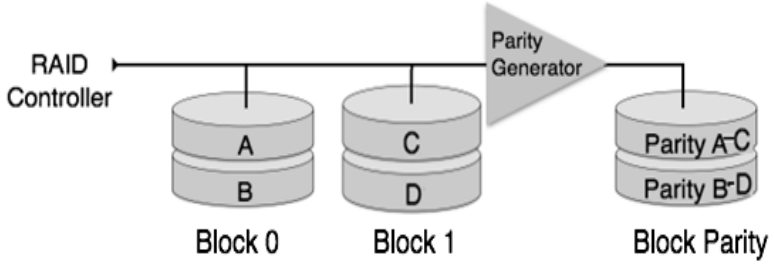
RAID 3 (Bit Interleaved Parity):

- RAID-3 **divide** the data into equal smaller blocks and stored onto **multiple disks**.
- The **parity bit** is generated for data and stored on a **different disk**.
- This technique makes it to overcome **single disk failures**.
- This parity information can be used to **recover the data on other disks.**
- This level uses **less storage space** than RAID 1 but the **parity disk** can become a **bottleneck**.



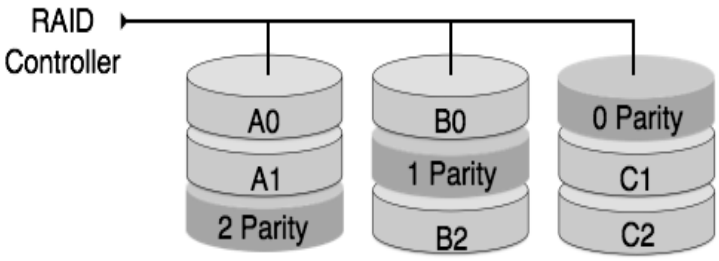
RAID 4 – Block Interleaved Parity:

- RAID-4 uses **large stripes,** which means you can read records from any single drive.
- An **entire block of data** is written onto data disks and then the parity is generated and stored on a different disk.
- Note that level 3 uses **byte-level striping**, whereas level 4 uses **block-level striping.**
- Both level 3 and level 4 require **at least three disks** to implement RAID.



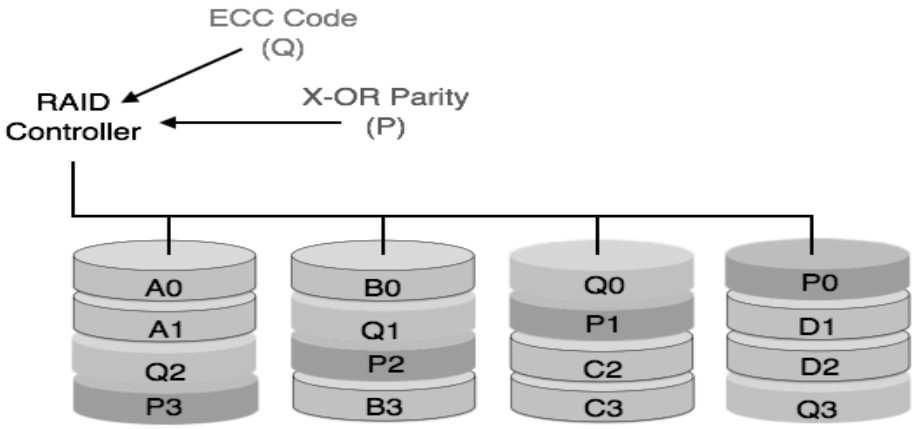
RAID 5 – Block-Interleaved Distributed Parity:

- RAID-5 writes whole **data blocks onto different disks**.
- The **parity** information is **striped across each drive**, allowing the array to function even if one drive were to fail.



RAID 6 – P+Q Redundancy:

- RAID-6 includes a **second parity scheme** that is distributed across the drives in the array.
- The use of **additional parity** allows the array to continue to function even if **<u>two disks fail</u>** simultaneously.
- This level requires at least **four disk drives** to implement RAID.



INDEXING:

» The data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

» Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

» The index is a type of data structure. It is used to locate and access the data in a database table quickly.

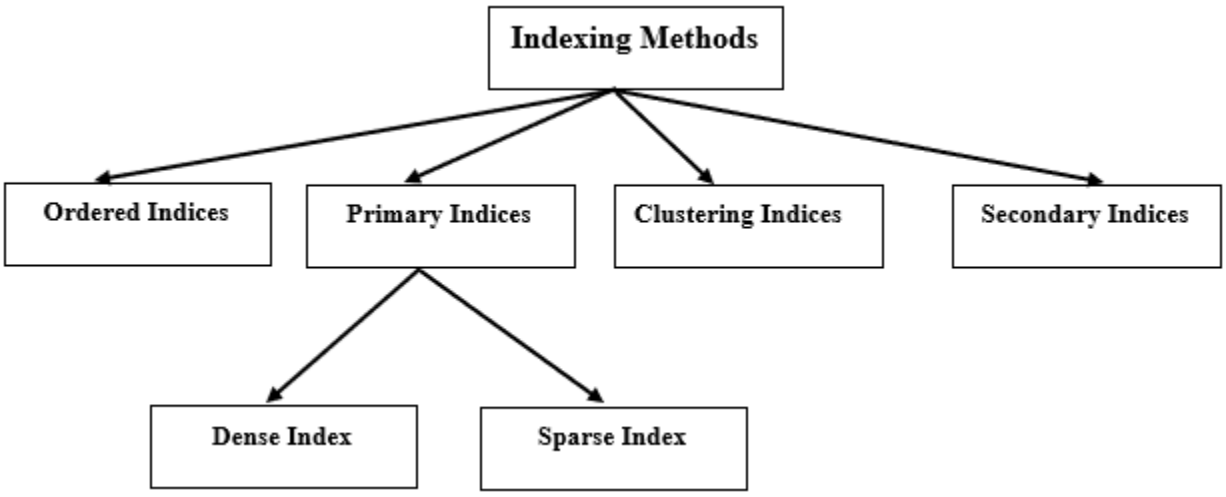
Index structure:

Search key	Data Reference

» The search key that contains a copy of the **primary key or candidate key** of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.

» The data reference, which contains a **set of pointers holding the address of the disk block** where the value of the particular key can be found.

» Indexing is defined based on its indexing attributes. Indexing can be of the following types –



1) Ordered indices:

» The indices are **usually sorted** to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of records and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

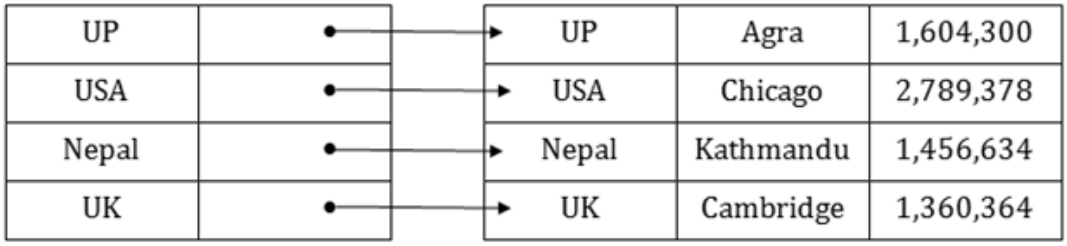
- ✓ In the case of a database **with no index**, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading **543*10=5430 bytes.**
- ✓ In the case of an **index**, we will search using indexes and the DBMS will read the record after reading **542*2= 1084** bytes which are very less compared to the previous case.

2) Primary Index:

- » If the index is created on the **basis of the primary key** of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- » As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- » The primary index can be classified into two types: **Dense index** and **Sparse index**.

a) Dense index

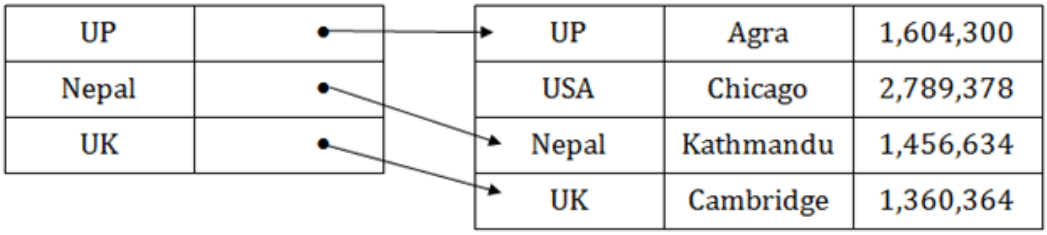
- » The dense index contains an **index record for every search key value** in the data file. It makes searching faster.
- » In this, the number of records in the index table is same as the number of records in the main table.
- » It **needs more space** to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



b) Sparse index

» In the data file, index record appears only for a few items. Each item points to a block.

» In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.

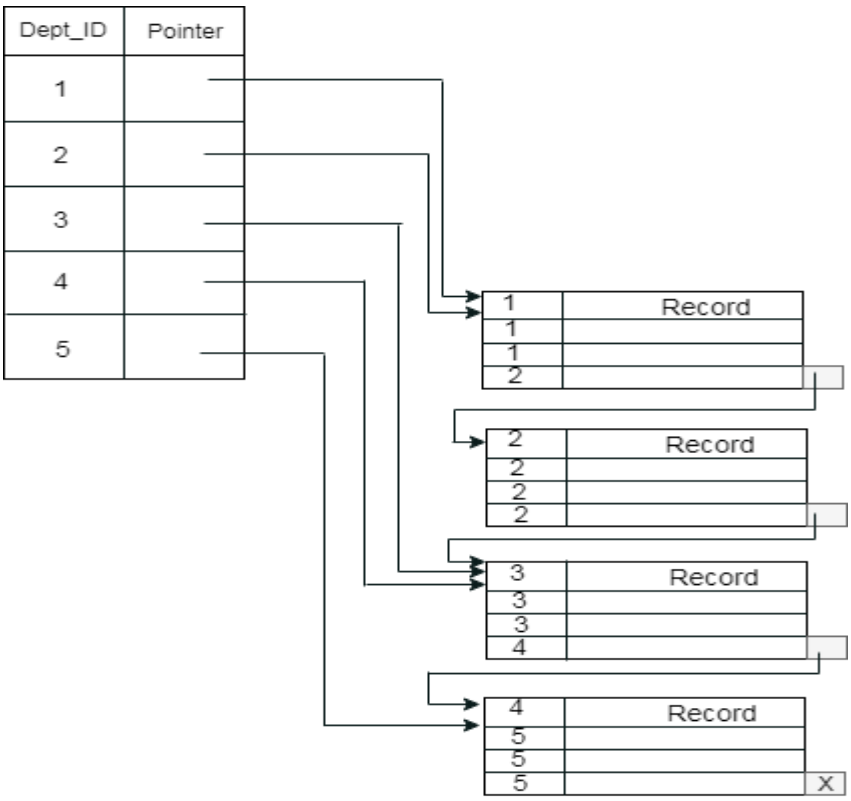


3) Clustering Index:

» A clustered index can be defined as an **ordered data file**. Sometimes the index is created on non-primary key columns which may not be unique for each record.

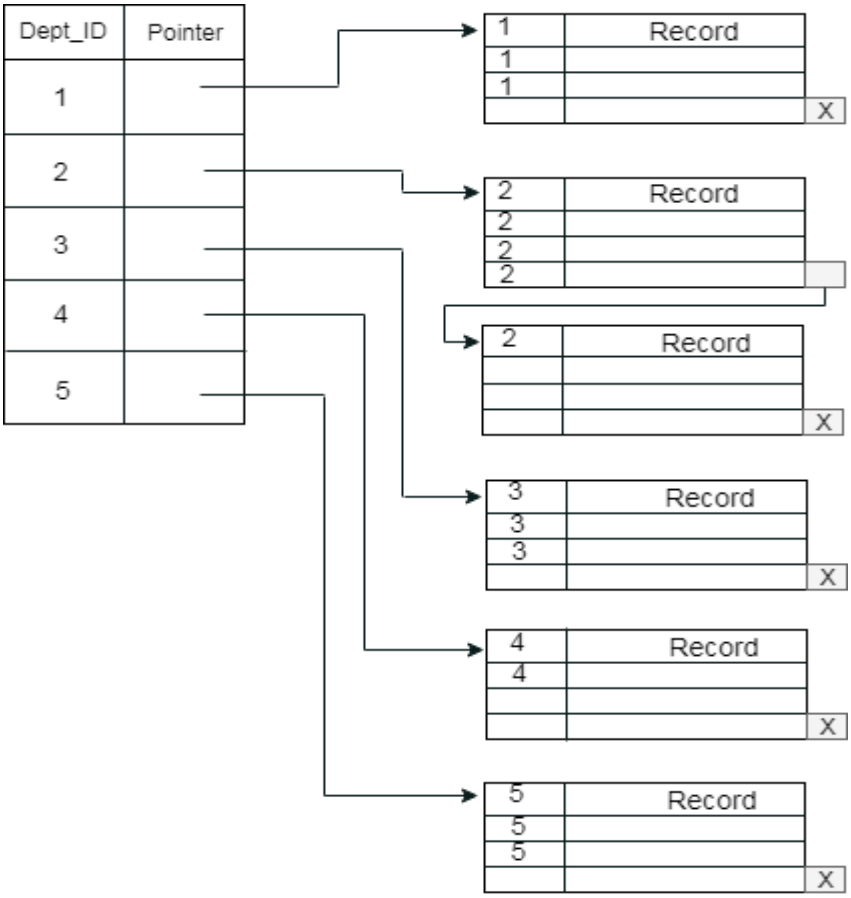
» In this case, to identify the record faster, we will **group two or more columns to get the unique value** and create index out of them. This method is called a clustering index.

» The records which have **similar characteristics are grouped**, and indexes are created for these group. **Example**:



» A company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_ID is a non-unique key.

» The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



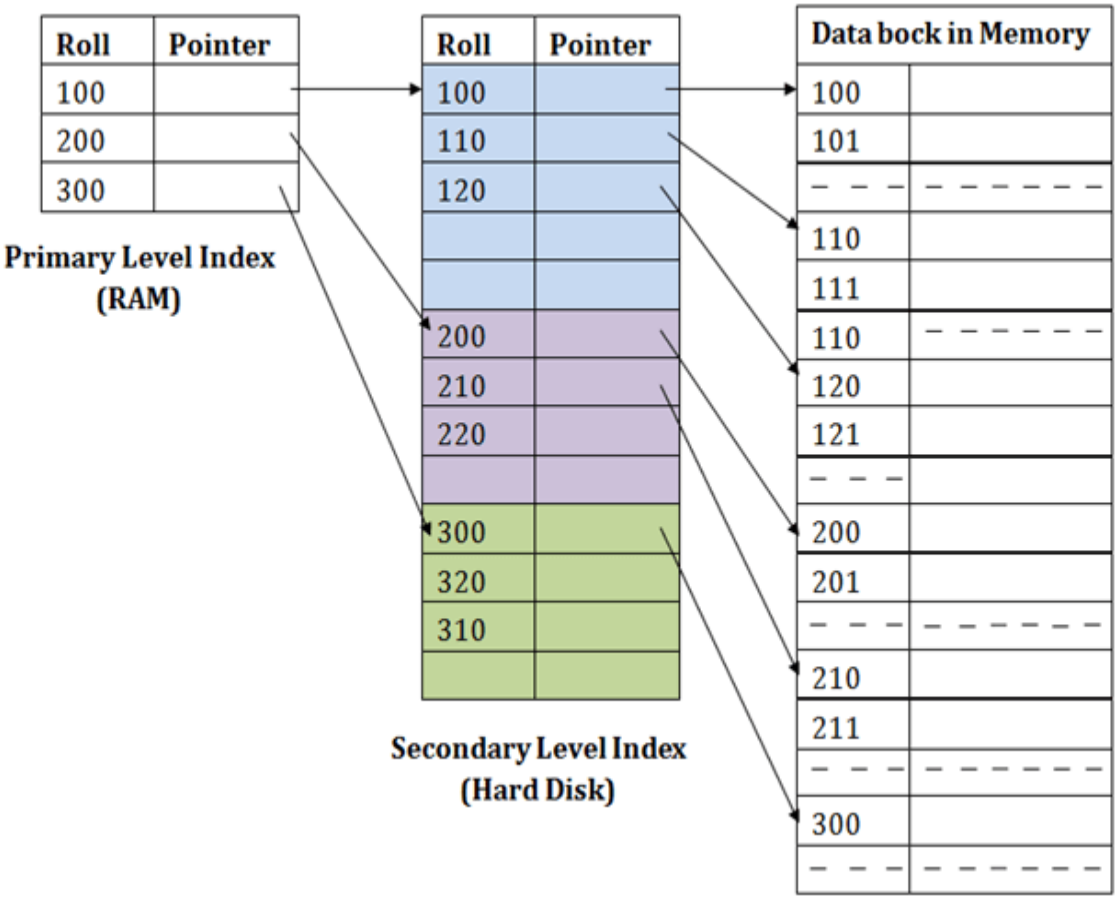
Secondary Index:

» In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster.

» Then the secondary memory searches the actual data based on the address got from mapping.

» If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

- In secondary indexing, to **reduce the size of mapping**, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).



B+ TREE:

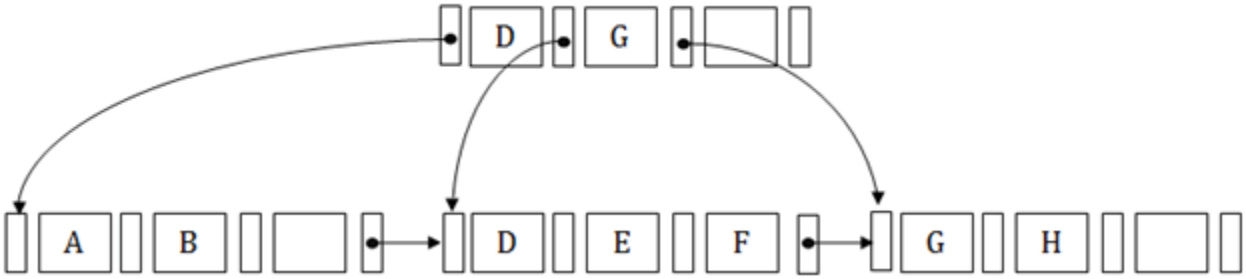
- The B+ tree is a **balanced binary search tree**. It follows a **multi-level index format**.
- In the B+ tree, **leaf nodes denote actual data pointers**. B+ tree ensures that all leaf nodes remain at the **same height**.

» In the B+ tree, the **leaf nodes are linked using a link list**. Therefore, a B+ tree can **support random access** as well as **sequential access**.

Structure of B+ Tree:

» In the B+ tree, **every leaf node is at equal distance** from the root node. The B+ tree is of the **order n** where **n is fixed for every B+ tree**.

» It contains an internal node and leaf node.



Internal node:

» An internal node of the B+ tree can contain at least n/2 record pointers except the root node.

» At most, an internal node of the tree contains n pointers.

Leaf node:

» The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.

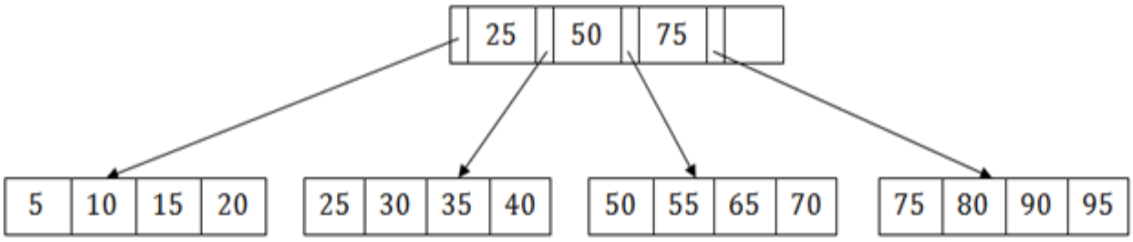
» At most, a leaf node contains n record pointer and n key values.

» Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree:

» Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

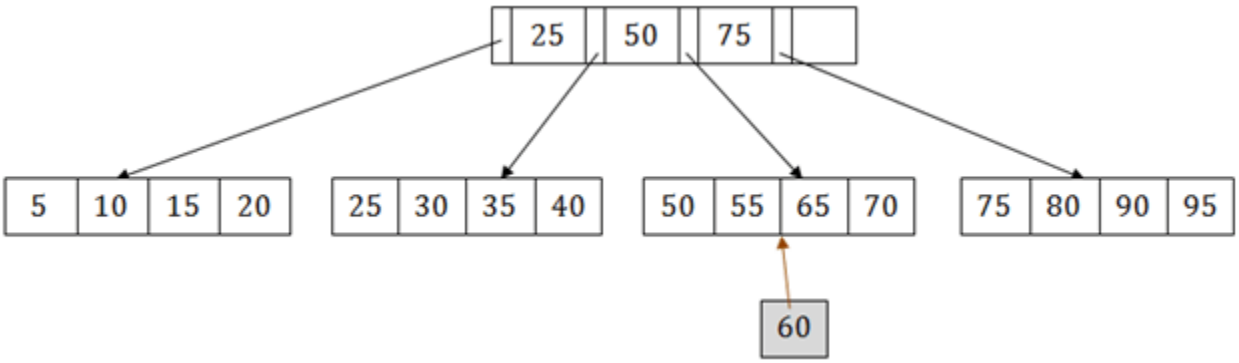
» So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



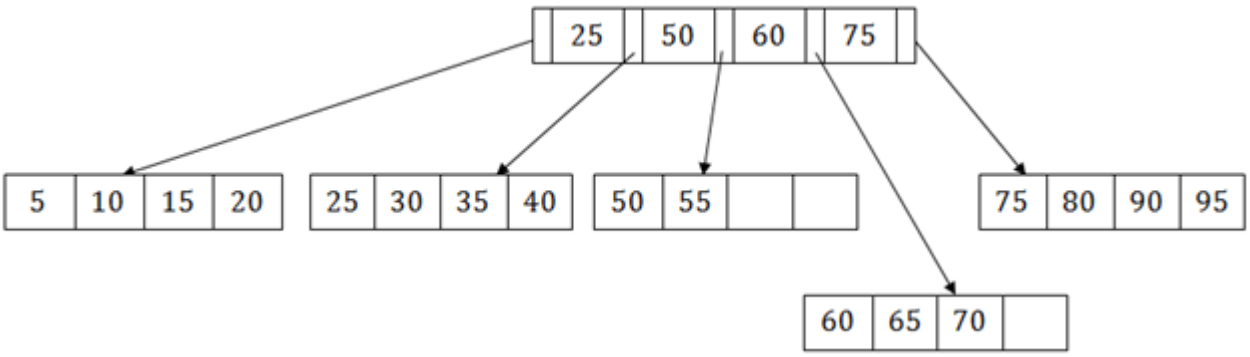
B+ Tree Insertion:

» Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

» In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



» The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So, we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.



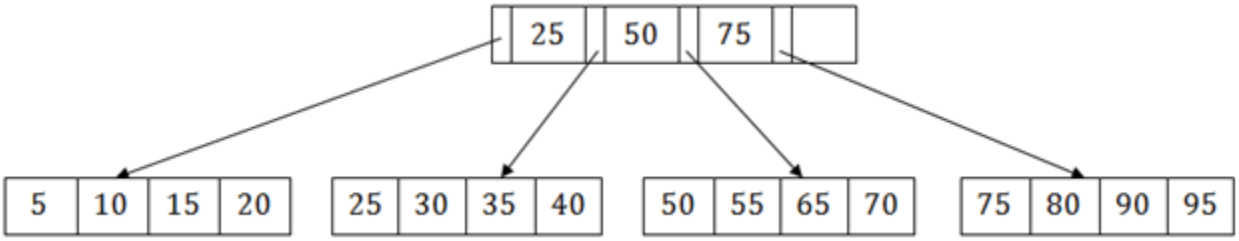
» If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

» This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion:

» Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

» After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:



HASHING:

» In a huge database structure, it is very inefficient to search all the index values and reach the desired data.

» Hashing is an effective technique to **calculate the direct location of a data record** on the disk without using index structure.

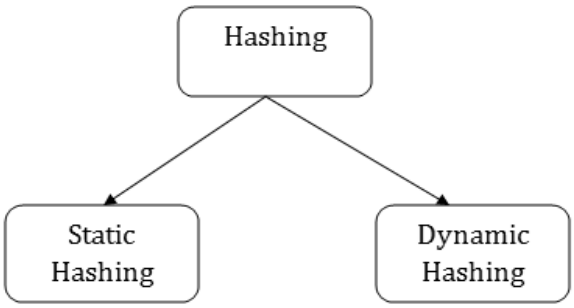
» Hashing uses **hash functions** with search keys as parameters to **generate the address of a data record**.

Hash Organization:

- **Bucket** − A hash file stores **data in bucket format**. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.

- **Hash Function** − A hash function, **h,** is a **mapping function** that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Types of Hashing:

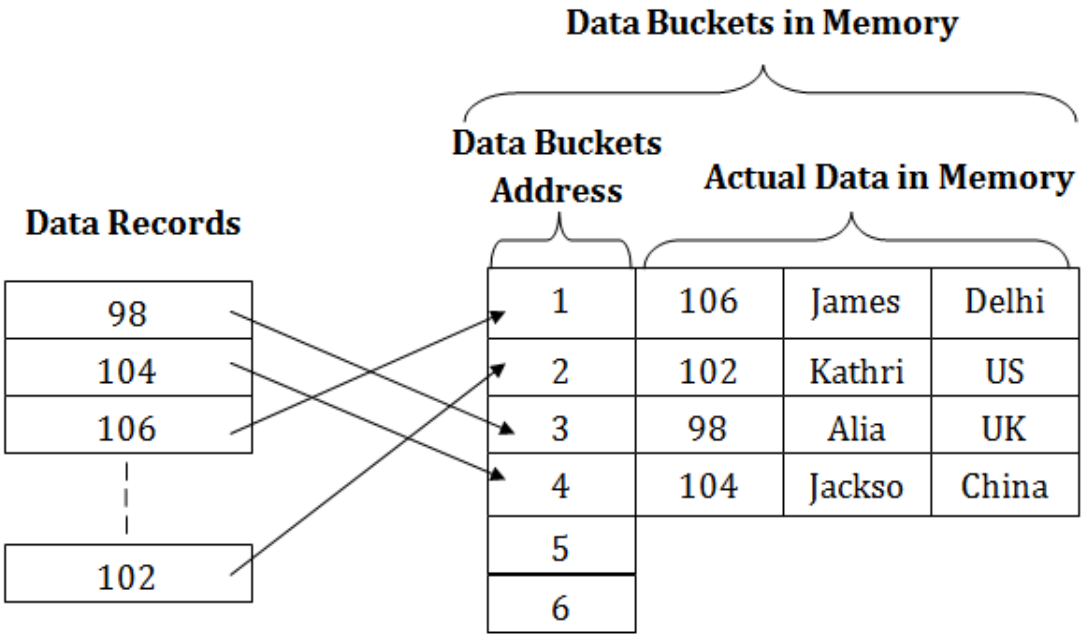


Static Hashing:

» In static hashing, the data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the **hash function mod (5)** then it will always result in same **bucket address 3**. Here, there will be no change in the bucket address.

» Hence in this static hashing, the number of data buckets in memory **remains constant** throughout.

» In this example, we will have **five data buckets** in the memory used to store the data.



Operations of Static Hashing:

1) **Searching a record**

   When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

2) **Insert a Record**

   When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

3) **Delete a Record**

   To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

4) **Update a Record**

   To update a record, we will first search it using a hash function, and then the data record is updated.

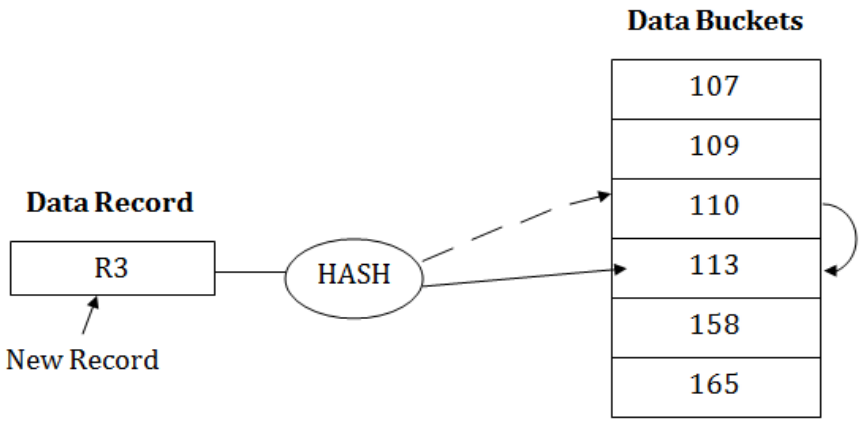
» If we want to **insert some new record into the file** but the address of a **data bucket** generated by the hash function **is not empty**, or **data already exists** in that address.

» This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

» To overcome this situation, there are various methods. Some commonly used methods are as follows:

1) Open Hashing:

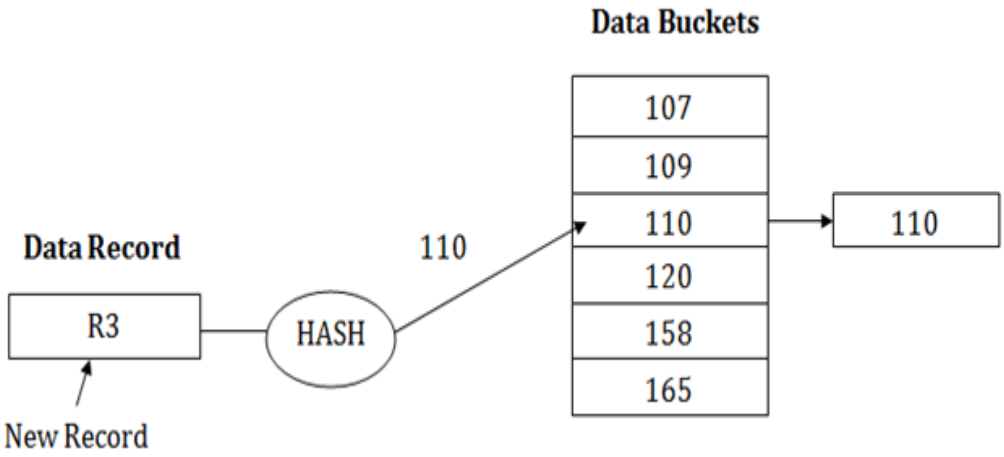
- » When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.
- » **For example:** suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



2) Close Hashing:

» When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

» **For example:** Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing:

» The dynamic hashing method is used to **overcome the problems of static hashing** like bucket overflow.

» In this method, **data buckets grow or shrink** as the records increases or decreases. This method is also known as **Extendable hashing method**.

» This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Search a key:

1. First, calculate the hash address of the key.
2. Check how many bits are used in the directory, and these bits are called as i.
3. Take the least significant i bits of the hash address. This gives an index of the directory.
4. Now using the index, go to the directory and find bucket address where the record might be.

Insert a new record:

1. Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
2. If there is still space in that bucket, then place the record in it.
3. If the bucket is full, then we will split the bucket and redistribute the records.

For example: Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

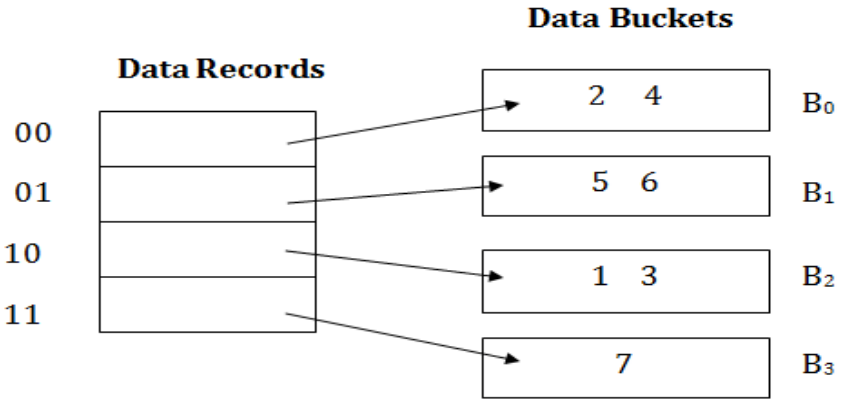
Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

» The last two bits of 2 and 4 are 00. So, it will go into bucket B0.

» The last two bits of 5 and 6 are 01, so it will go into bucket B1.

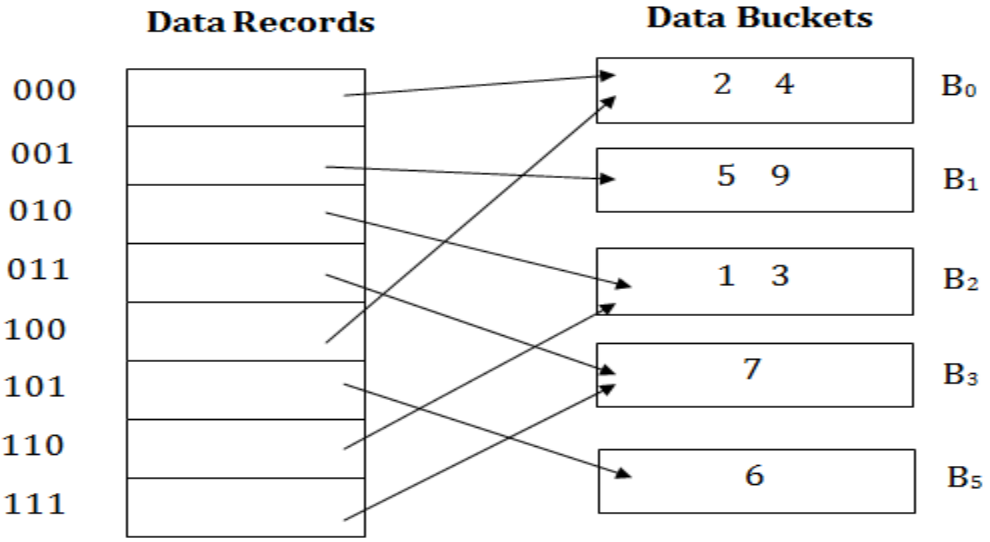
» The last two bits of 1 and 3 are 10, so it will go into bucket B2.

» The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing:

1. In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
2. In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
3. This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing:

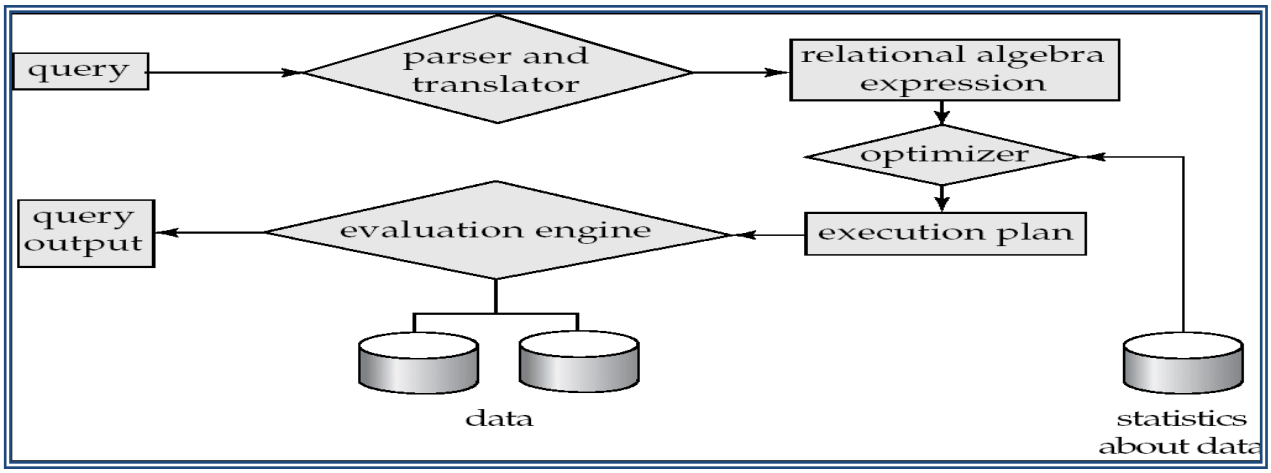
1. In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
2. In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

QUERY PROCESSING:

Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database language into expressions that can be used at physical level of the file system, a variety of query-optimizing and actual evaluation queries.

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Parsing and translation:

1. Translate the query into its internal form. This is then translated into Relational Algebra.
2. Parser checks syntax, verifies relations

   A relational algebra expression may have many equivalent expressions

   E.g., $\sigma_{balance<2500}(\prod_{balance}(account))$ is equivalent to $\prod_{balance}(\sigma_{balance<2500}(account))$

Evaluation:

1. The **Query-Execution Engine** takes a query-evaluation plan, executes that plan, and returns the answers to the query.
2. Each Relational Algebra operation can be evaluated using one of several different algorithms correspondingly, a relational-algebra expression can be evaluated in many ways.
3. Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.

   Example: can use an index on *balance* to find accounts with balance < 2500,

   (or)

   can perform complete relation scan and discard accounts with balance ≥ 2500

$\prod$ **balance**

$\sigma_{balance<2500}$

account

Query Optimization:

- Amongst all equivalent evaluation plans choose the one with lowest cost.
- Cost is estimated using statistical information from the database catalog

Example: Number of tuples in each relation, size of tuples, etc.

Measures of Query Cost:

- Cost is generally measured as total elapsed time for answering query
- Many factors contribute to time cost –
  - *disk accesses, CPU,* or even network *communication*
- Typically, *disk access* is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  1. Number of seeks * average-seek-cost
  2. Number of blocks read * average-block-read-cost
  3. Number of blocks written * average-block-write-cost

Note:

- ❖ Cost to write a block is greater than cost to read a block.
- ❖ Data is read back after being written to ensure that the write was successful

- For simplicity we just use the *number of block transfers from disk and the number of seeks* as the cost measures
  1. t_T – time to transfer one block
  2. t_S – time for one seek
  3. Cost for b block transfers plus S seeks
     $b * t_T + S * t_S$
- We ignore CPU costs for simplicity
  1. Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae
- Several algorithms can reduce disk IO by using extra buffer space
  1. Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
     - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  1. But hard to take into account for cost estimation

Selection Operation:

- **File scan** – search algorithms that locate and retrieve records that fulfill a selection condition.

Algorithm A1 (*linear search*): Scan each file block and test all records to see whether they satisfy the selection condition.

2. Cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation r
3. If selection is on a key attribute, can stop on finding record
    - cost = (b_r /2) block transfers + 1 seek
4. Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices

Algorithm A2 (*binary search*): Applicable if selection is an equality comparison on the attribute on which file is ordered.

5. Assume that the blocks of a relation are stored contiguously
6. Cost estimate (number of disk blocks to be scanned):
    - cost of locating the first tuple by a binary search on the blocks
        - $\lceil \log_2(b_r) \rceil * (t_T + t_S)$
    - If there are multiple records satisfying selection
        - *Add transfer cost of the* number of blocks containing records that satisfy selection condition

QUERY OPTIMIZATION:

- **Query optimization** is the part of the **query** process in which the database system compares different **query** strategies and chooses the one with the least expected cost.
- The **query optimizer**, which carries out this function, is a key part of the relational database and determines the most efficient way to access data.
- Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 − Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas −

Table Name: EMPLOYEE

EmpID	EName	Salary	DeptNo	DateOfJoining

Table Name: DEPARTMENT

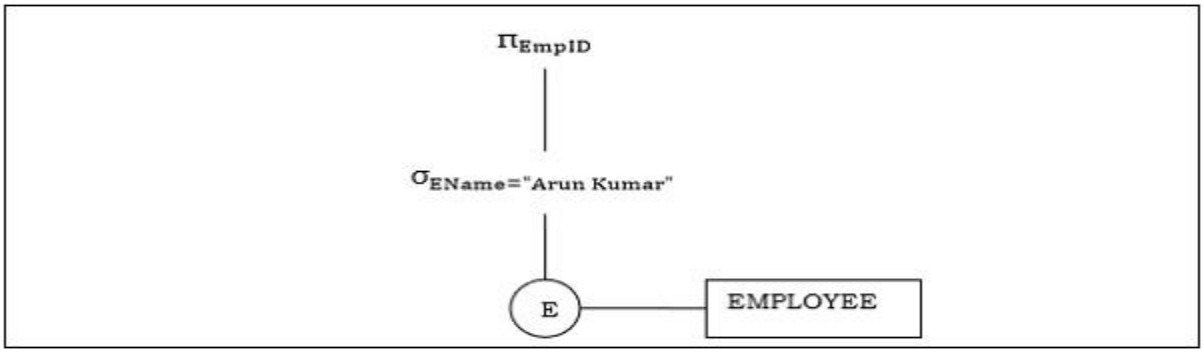
DNo	DName	Location

Example:1

Let us consider the query as the following.

$\Pi_{EmpID}(\sigma_{(EName = ArunKumar)}(EMPLOYEE)$

The corresponding query tree will be −

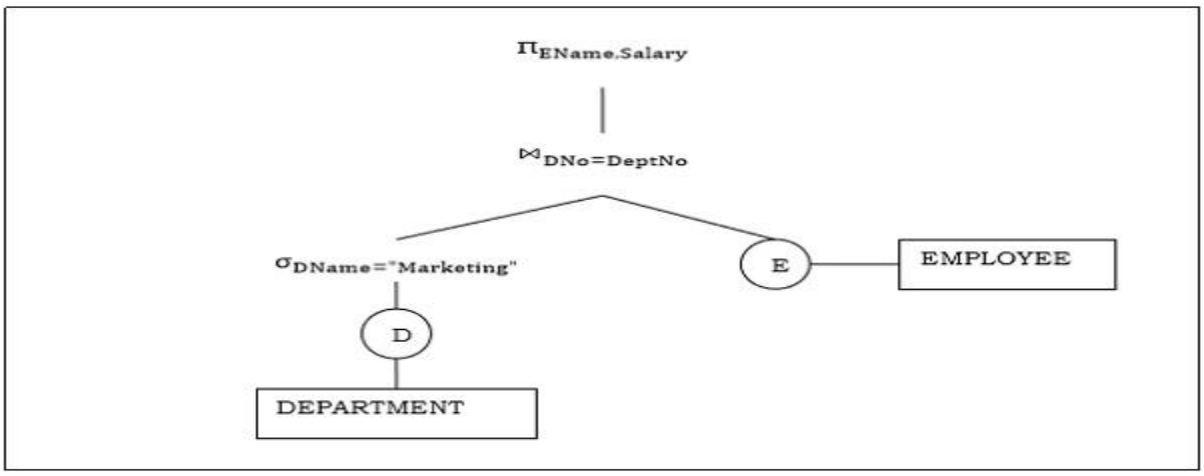


Example: 2

Let us consider another query involving a join.

$\Pi_{(EName, Salary)}(\sigma_{(DName="Marketing")}(DEPARTMENT)) \bowtie (\sigma_{(DNo=DeptNo)})(EMPLOYEE)$

Following is the query tree for the above query.



Step 2 − Query Plan Generation

- After the query tree is generated, a **query plan** is made. A query plan is an extended query tree that includes **access paths for all operations** in the query tree.
- Access paths specify **how the relational operations in the tree should be performed**. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.
- Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3− Code Generation

- Code generation is the **final step in query optimization**. It is the **executable form of the query**; whose form depends upon the type of the underlying operating system.
- Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization:

Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

1) Exhaustive Search Optimization:

In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and

space complexity owing to the large solution space. For example, dynamic programming technique.

2) Heuristic Based Optimization:

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

UNIT IV TRANSACTION AND CONCURRENCY

UNIT-IV TRANSACTIONS 9

Transaction Concept-A Simple Transaction Model-Storage Structure-Transaction Atomicity and Durability-Transaction Isolation- Transaction Isolation Levels- Serializability- Transactions as SQL Statements-Concurrency Control- Lock-Based Protocols- Time Stamp-Based Protocols-Validation Based Protocols- Database recovery - Deadlock Handling.

1. TRANSACTION:

Definition:

"A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database."

Or

"Transaction is a set of operations which are all logically related."

Or

"Transaction is a single logical unit of work formed by a set of operations."

1.1 Operations in Transaction:

The main operations in a transaction are-

- Read Operation
- Write Operation

1. Read Operation: Read operation reads the data from the database and then stores it in the buffer in main memory.

For example: - **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

2. Write Operation: Write operation writes the updated data value back to the database from the buffer.

For example: - **Write(A)** will write the updated value of A from the buffer to the database.

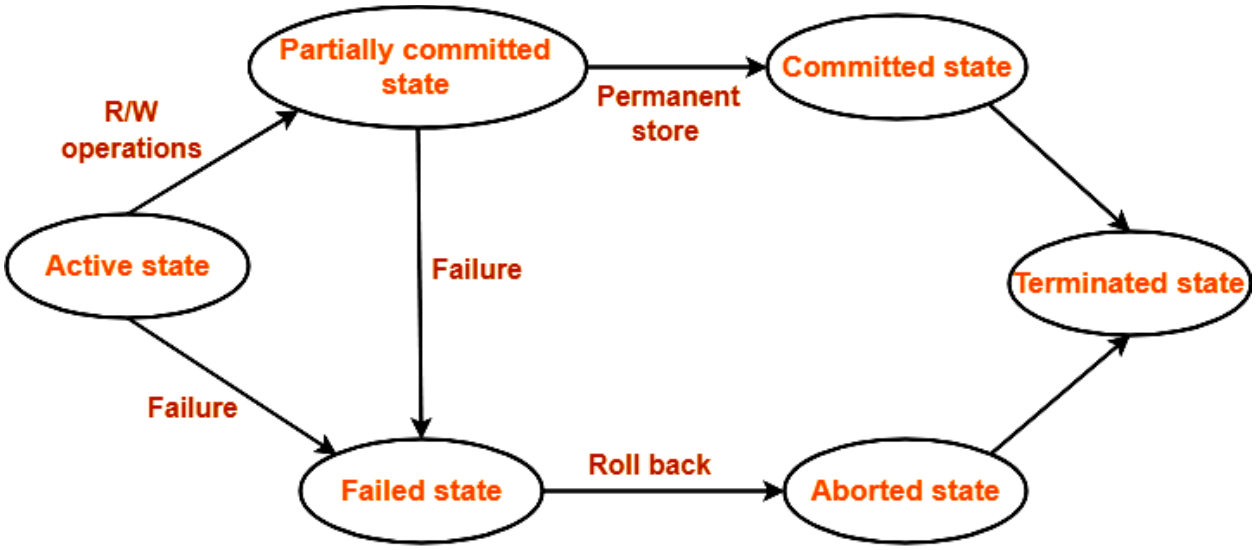
1.2 Transaction States:

A transaction goes through many different states throughout its life cycle.

These states are called as **transaction states**.

Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state



Transaction States in DBMS

1.2.1. Active State:

- This is the first state in the life cycle of a transaction.
- A transaction is called in an **active state** as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

1.2.2. Partially Committed State:

- After the last instruction of transaction has executed, it enters into a partially committed state.
- After entering this state, the transaction is considered to be partially committed.

- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

1.2.3. Committed State:

- After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
- Now, the transaction is considered to be fully committed.

Key Notes:

- After a transaction has entered the committed state, it is not possible to roll back the transaction.
- In other words, it is not possible to undo the changes that have been made by the transaction.
- This is because the system is updated into a new consistent state.
- The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operations.

1.2.4. Failed State:

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

1.2.**5. Aborted State:**

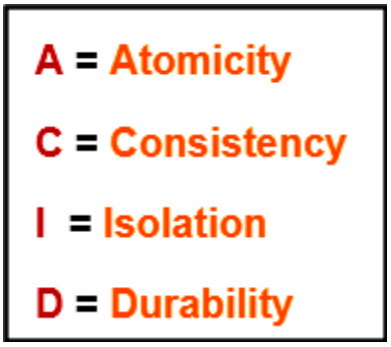
- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state.

1.2.**6. Terminated State:**

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

1.3 ACID Properties:

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as ACID Properties of a transaction.



1.3.1. Atomicity:

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as "**All or nothing rule**".
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

1.3.2. Consistency:

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

1.3.3. Isolation:

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.

- Changes made by a transaction become visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

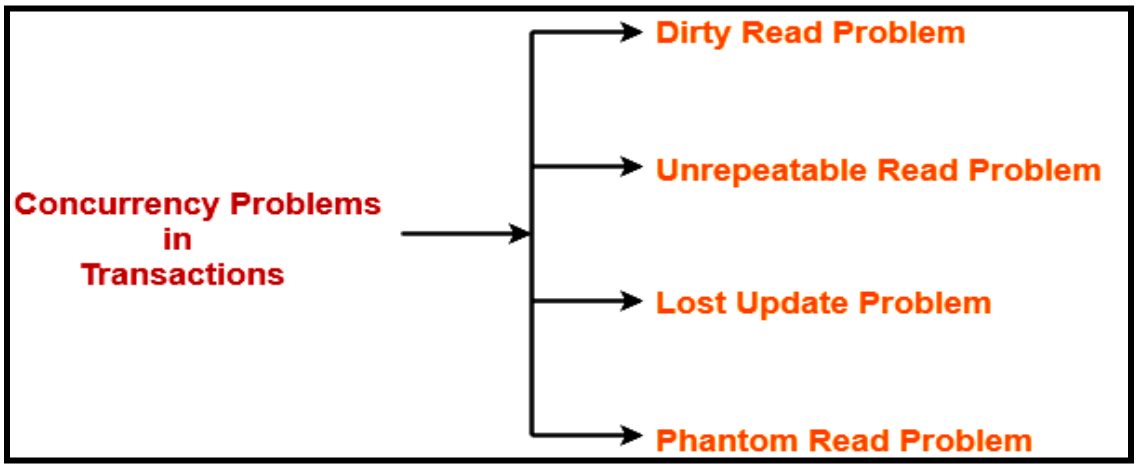
1.3.4. Durability:

- This property ensures that all the changes made by a transaction after their successful executions are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there any failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

1.4 Concurrency Problems

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as **concurrency problems**.

The concurrency problems are: -



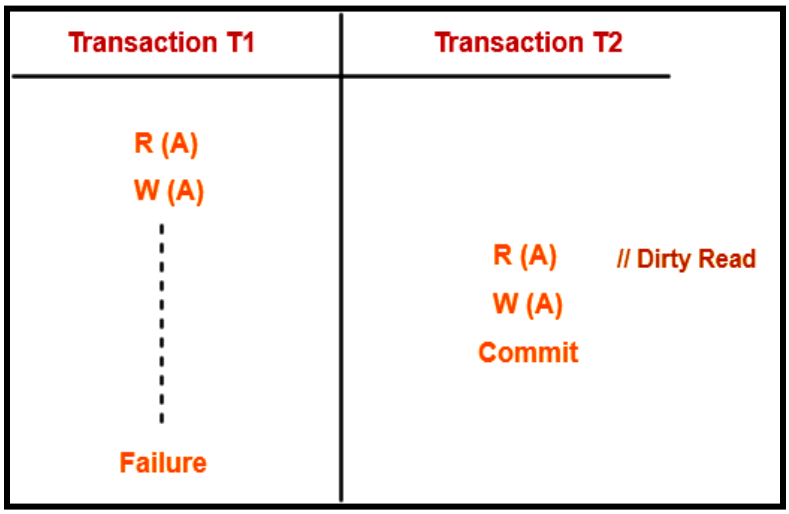
1.4.1. Dirty Read Problem: -

"Reading the data written by an uncommitted transaction is called as dirty read."

This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.

- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.



Here,

1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

1. T2 reads the dirty value of A written by the uncommitted transaction T1.
2. T1 fails in later stages and roll backs.
3. Thus, the value that T2 read now stands to be incorrect.
4. Therefore, database becomes inconsistent.

1.4.2. Unrepeatable Read Problem: -

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Transaction T1	Transaction T2
R (X)	
	R (X)
W (X)	
	R (X) // Unrepeated Read

Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

In this example,

1. T2 gets to read a different value of X in its second reading.
2. T2 wonders how the value of X got changed because according to it, it is running in isolation.

1.4.3. Lost Update Problem: -

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Transaction T1	Transaction T2
R (A)	
W (A)	
⋮	W (A)
	Commit
Commit	

Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.

3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

1. T1 writes the over written value of X in the database.
2. Thus, update from T1 gets lost.

1.4.4. Phantom Read Problem:

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later; it finds that the variable does not exist.

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

1. T2 finds that there does not exist any variable X when it tries reading X again.
2. T2 wonders who deleted the variable X because according to it, it is running in

Avoiding Concurrency Problems: -

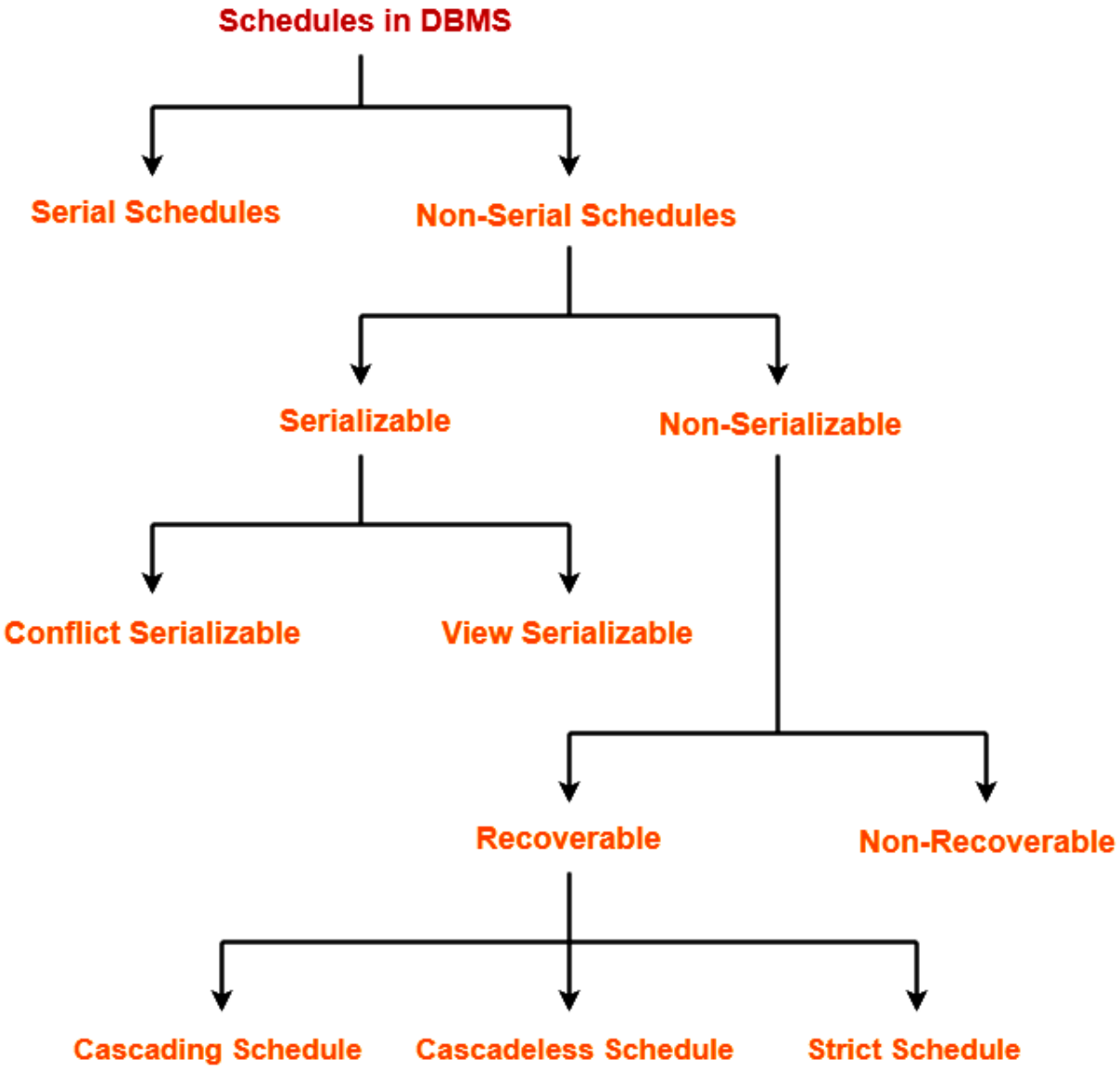
- To ensure consistency of the database, it is very important to prevent the occurrence of above problems.
- **Concurrency Control Protocols** help to prevent the occurrence of above problems and maintain the consistency of the database.

2. SCHEDULES: -

Definition: **"***The order in which the operations of multiple transactions appear for execution is called as a schedule."*

2.1 Types of Schedules:

In DBMS, schedules may be classified as-



2.2 Serial Schedules: -

In serial schedules,

- All the transactions execute serially one after the other.

- When one transaction executes, no other transaction is allowed to execute.

Characteristics-

Serial schedules are always-

1. Consistent
2. Recoverable
3. Cascadeless
4. Strict

Example:

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a Serial Schedule.

2.3 Non-Serial Schedules:

In non-serial schedules,

- Multiple transactions execute concurrently.
- Operations of all the transactions are interleaved or mixed with each other.

Characteristics-

Non-serial schedules are **NOT** always-

1. Consistent
2. Recoverable

3. Cascadeless
4. Strict

Example-1:

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.

Example-2:

Transaction T1	Transaction T2
	R (A)
R (A)	
W (B)	
	R (B)
	Commit
R (B)	
W (B)	
Commit	

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.

SERIALIZABILITY: -

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

2.3.1 SERIALIZABLE SCHEDULES

- If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a ***serializable schedule***.

Characteristics-

Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

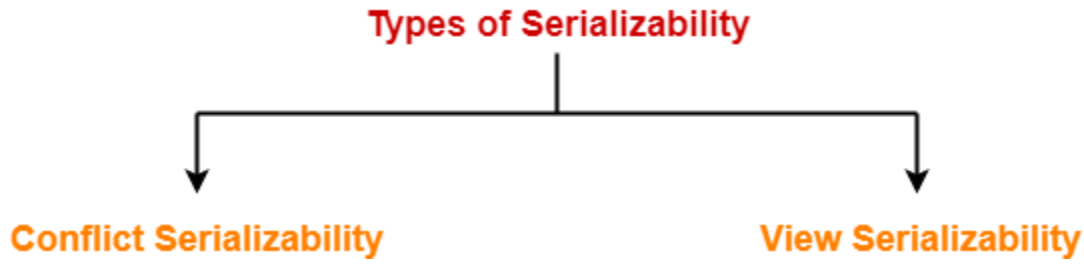
Serial Schedules Vs Serializable Schedules

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)	Serializable Schedules are always better than serial schedules. (due to above reason)

Types of Serializability: -

Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability



2.3.1.1 CONFLICT SERIALIZABILITY: -

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.

Conflicting Operations-

- Two operations are called as conflicting operations if all the following conditions hold true for them-
  1. Both the operations belong to different transactions
  2. Both the operations are on the same data item
  3. At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01: Find and list all the conflicting operations.

Step-02: Start creating a precedence graph by drawing one node for each transaction.

Step-03: # Draw an edge for each conflict pair such that if Xi (V) and Y_j (V) forms a conflict pair then draw an edge from Ti to Tj.

This ensures that Ti gets executed before Tj.

Step-04: Check if there is any cycle formed in the graph.

If there is no cycle found, then the schedule is conflict serializable otherwise not.

Practice Problems Based On Conflict Serializability-

Problem-01: Check whether the given schedule S is conflict serializable or not-

S : R1(A) , R2(A) , R1(B) , R2(B) , R3(B) , W1(A) , W2(B)

Solution-

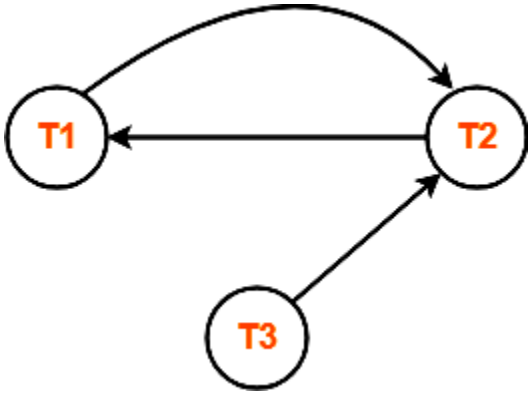
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Problem-02: Check whether the given schedule S is conflict serializable and recoverable or not-

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Solution-

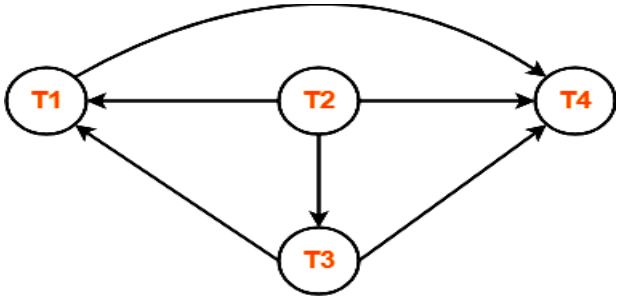
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- R2(X) , W3(X) (T2 → T3)
- R2(X) , W1(X) (T2 → T1)
- W3(X) , W1(X) (T3 → T1)
- W3(X) , R4(X) (T3 → T4)
- W1(X) , R4(X) (T1 → T4)
- W2(Y) , R4(Y) (T2 → T4)

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

2.3.1.2 VIEW SERIALIZABILITY: -

Definition: "*If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule."*

View Equivalent Schedules-

- Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.
- Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

- For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Condition-02:

- If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Condition-03:

- For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Checking Whether a Schedule is View Serializable or Not-

Method-01:

- Check whether the given schedule is conflict serializable or not.
- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable.

Key Points:

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

- Check if there exists any blind write operation.
  *(Writing without reading is called as a **blind write**).*
- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there any blind write, then the schedule may or may not be view serializable.

Key Points:

- *No blind write means not a view serializable schedule.*

Method-03:

1. In this method, try finding a view equivalent serial schedule.
2. Then, draw a graph using those dependencies.
3. If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

Checking Whether a Schedule is View Serializable Or Not-

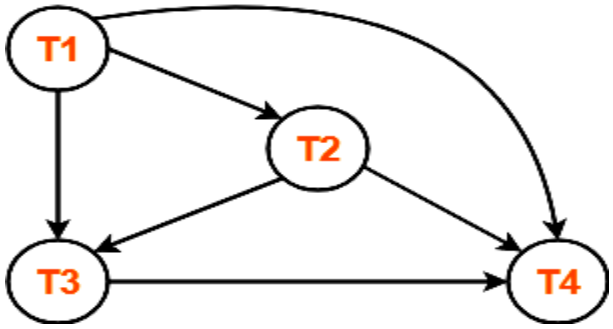
Problem-01:

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)

- If a schedule is conflict serializable, then it is surely view serializable.
- List all the conflicting operations and determine the dependency between the transactions-
  - W1(B) , W2(B) (T1 → T2)
  - W1(B) , W3(B) (T1 → T3)
  - W1(B) , W4(B) (T1 → T4)

- W2(B) , W3(B) (T2 → T3)
- W2(B) , W4(B) (T2 → T4)
- W3(B) , W4(B) (T3 → T4)

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

2.3.2 NON SERILIAZABLE SCHEDULE:

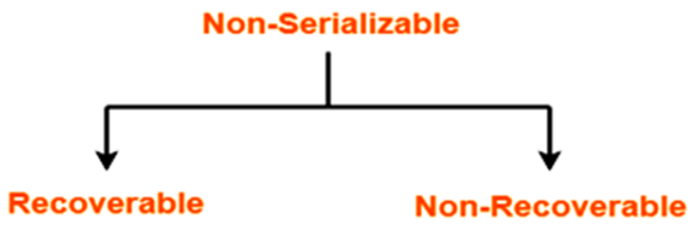
Definition:

"A non-serial schedule which is not serializable is called as a non-serializable schedule."

Characteristics-

- May or may not be consistent
- May or may not be recoverable

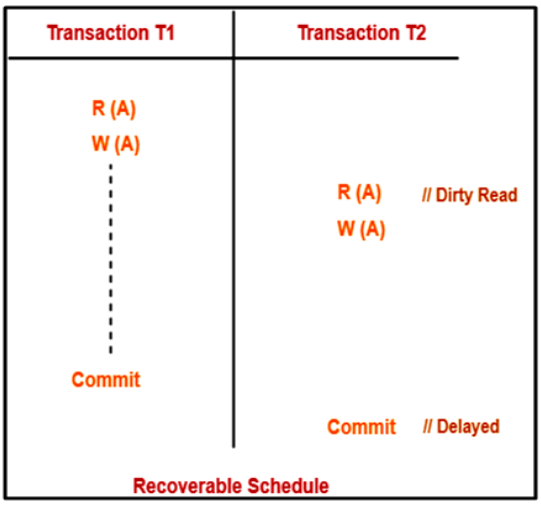
Types of Non-Serializable Schedule:



2.3.2.1 RECOVERABLE SCHEDULE:

If in a schedule,

1. A transaction performs a dirty read operation from an uncommitted transaction.
2. And its commit operation is delayed till the uncommitted transaction commits or roll backs, and then such a schedule is known as a ***Recoverable Schedule***.



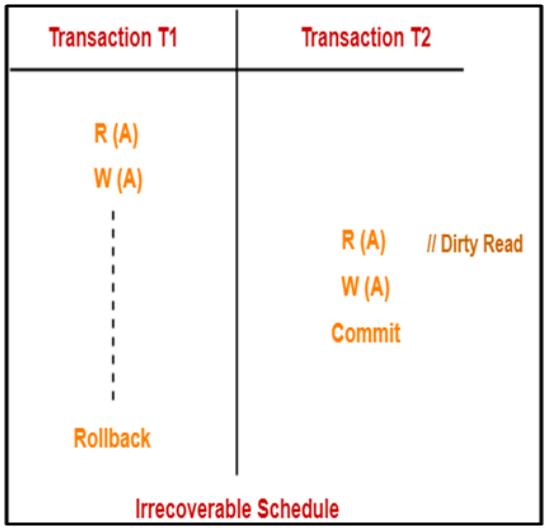
Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

2.3.2.2 NON -RECOVERABLE SCHEDULE:

If in a schedule,

1. A transaction performs a dirty read operation from an uncommitted transaction
2. And commits its operations before the transaction from which it has read the value then such a schedule is known as an ***Non - Recoverable Schedule.***



Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Checking Whether a Schedule is Recoverable or Irrecoverable:

Method-01:

- Check whether the given schedule is conflict serializable or not.
- If the given schedule is conflict serializable, then it is surely recoverable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be recoverable. Go and check using other methods.

Key Points:

- All conflict serializable schedules are recoverable.
- All recoverable schedules may or may not be conflict serializable.

Method-02:

- Check if there exists any dirty read operation.
  *(Reading from an uncommitted transaction is called as a **dirty read**)*
- If there does not exist any dirty read operation, then the schedule is surely recoverable. Stop and report your answer.
- If there any dirty read operation, then the schedule may or may not be recoverable.

Checking Whether a Schedule is Recoverable or Irrecoverable:

Problem: 01

SCHEDULE B

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
	Commit
Abort	

- The Schedule B is Not Recoverable.
- T2 reads item X from T1.
- T2 commits before T1 commits.
- T1 Aborts after T2 Commits.
- The value of X that T2 read is no longer valid.
- T2 must be Aborted after its Committed.
- Leading to schedule that is not recoverable.

Key Points:

- For the schedule to be recoverable, the T2 Commit operation in Schedule B must be postponed until T1 Commits.

Problem: 02

SCHEDULE C

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
Write(Y)	
Commit	
	Commit

- The Schedule C is Recoverable.
- T2 reads item X from T1.
- T2 commits after T1 commits.

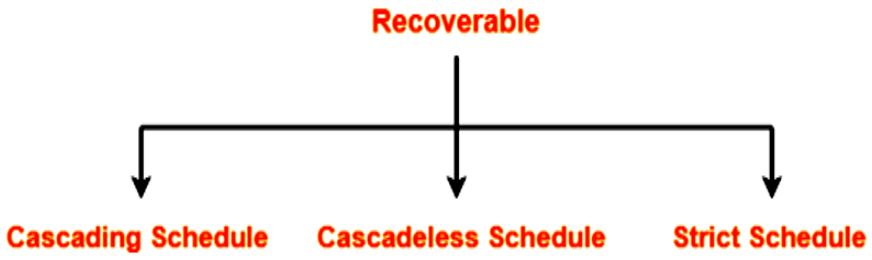
Problem: 03

SCHEDULE D

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
	Write(X)
Write(Y)	
Abort	
	Abort

- The Schedule D is Recoverable.
- T2 reads item X from T1.
- T1 Aborts instead of Committing, Then T2 Should also Abort.
- Because the value of X it read is no longer valid

2.3.1.1 TYPES OF RECOVERABLE SCHEDULE:

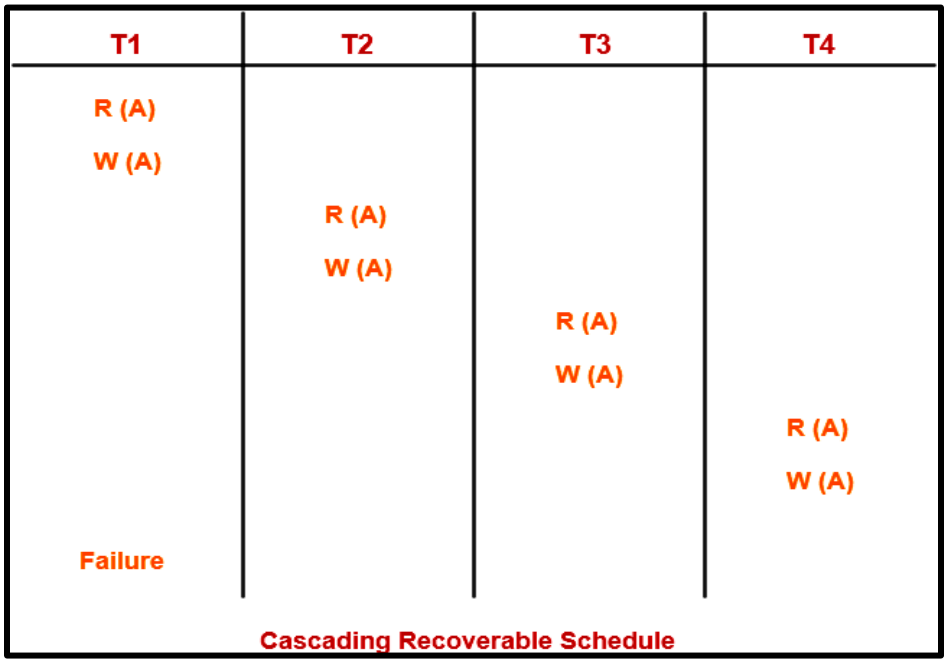


I. Cascading Schedule:

If in a schedule,

1. Failure of one transaction causes several other dependent transactions to rollback or abort, and then such a schedule is called as a ***Cascading Schedule*** or ***Cascading Rollback*** or ***Cascading Abort***.
2. It simply leads to the wastage of CPU time.

Example:



Hère,

- Transaction T2 dépends on transaction T1.
- Transaction T3 dépends on transaction T2.
- Transaction T4 dépends on transaction T3.

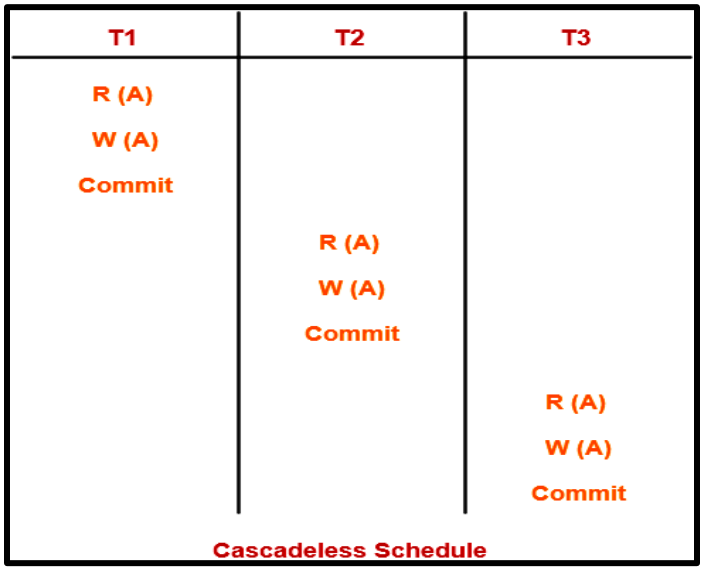
In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.

Such a rollback is called as a **Cascading Rollback**.

II. Cascadeless Schedule-

- If in a schedule,
- "A transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called As a Cascadeless Schedule."
- In other words, ***"Cascadeless schedule allows only committed read operations."***
- Therefore, it avoids cascading roll back and thus saves CPU time.



- Cascadeless schedule allows only committed read operations.
- However, it allows uncommitted write operations.

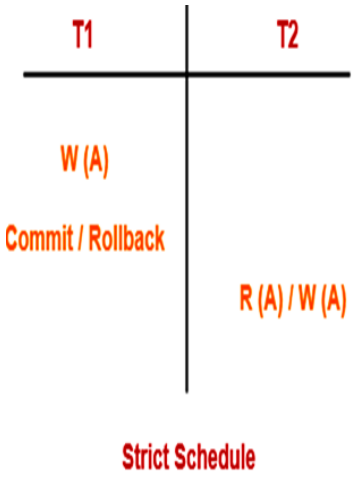
III. Strict Schedule-

If in a schedule,

- "A transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, and then such a schedule is called as a **Strict Schedule**."

In other words,

- "Strict schedule allows **only committed read and write** operations."
- Strict schedule implements more restrictions than Cascadeless schedule.



Remember-

- Strict schedules are more strict than Cascadeless schedules.
- All strict schedules are Cascadeless schedules.
- All Cascadeless schedules are not strict schedules.

3. CONCURRENCY CONTROL:

- Concurrency control techniques that are used to ensure the non-interference or Isolation property of concurrently executing transactions.
- There are **three methods for concurrency control**. They are as Follows:

1. Locking Methods
2. Time-stamp Methods
3. Optimistic Methods

1. Locking Methods of Concurrency Control:

- "A lock is a variable, associated with the data item, which controls the access of that data item."
- Locking is the most widely used form of the concurrency control mechanism.
- Locks are further divided into three fields:

a) Lock Granularity
b) Lock Types
c) Deadlocks

a) Lock Granularity:

- A database is a collection of named data items.
- The size of the data item chosen as the unit of protection by a concurrency control program is called **GRANULARITY**.

- Locking can take place at the following level :
    1. Database level locking.
    2. Table level locking.
    3. Page level locking.
    4. Row (Tuple) level locking.
    5. Attributes (fields) level locking.

b) Lock Types:

The DBMS mainly uses following types of locking techniques.

1) Binary Locking
2) Shared / Exclusive Locking
3) Two - Phase Locking (2PL)

1) **BINARY LOCKS**:

- A binary lock can have two states or values:
    1) Locked state - (1)
    2) Unlocked state - (0)
- If the value of the lock on X is 1, then the item X cannot be accessed by a other operation.
- If the value of the lock on X is 0, the item can be accessed by other operation.
- The current state of the lock associated with item X is **LOCK(X).**
- Two operations, ***lock_item*** and ***unlock_item***, are used with binary locking.

Lock_item(X):

- A transaction requests an item.
- If LOCK(X) = 1, then transaction has to **wait**.
- If LOCK(X) = 0, it is set to 1 and the transaction is allowed to access item X.

Unlock_item (X):

- Sets LOCK(X) to 0 (unlocks) so that X may be accessed by other transactions. (LOCK(X)←0)
- A binary lock enforces mutual exclusion on the data item ; i.e., at a time only one transaction can hold a lock.

Binary Lock Rules:

1. A transaction T must issue the operation ***lock_item(X)*** before any ***read_item(X)*** or ***write_item(X)*** operations are performed in T.
2. A transaction T must issue the operation ***unlock_item(X)*** after all ***read_item(X)*** and write_item(X) operations are completed in T.
3. A transaction T will not issue a ***lock_item(X)*** operation if it already holds the lock on item X.
4. A transaction T will not issue an ***unlock_item(X)*** operation unless it already holds the lock on item X.

2) **SHARED / EXCLUSIVE LOCK:**

- Binary locking scheme allows at most one transaction can hold a lock on a given time.
- We should allow several transactions to access the same data item X if they all access X for reading purposes only.
- If a transaction is to write a data item X, it must have exclusive access to X.
- For this purpose, a different type of lock called multiple-mode lock is used.
- This scheme is called shared/exclusive lock.
- There are three locking operations:

  1. Read_lock(X) ***2. Write_lock(X)*** ***3.Unlock(X)***

- A lock associated with an data item X, **LOCK(X)**
- Has three states – 1) read-locked state
  2) write-locked state
  3) unlocked state.
- A read-locked item is also called share-locked because other transactions are allowed to read the item.
- A write-locked item is called exclusive –locked because a single transaction exclusively holds the lock on the item.

Shared Lock:

- These locks are referred as read locks, and denoted by **'S'.**
- If a transaction T has obtained Shared-lock on data item X, then T can read X, but cannot write X.
- Multiple Shared lock can be placed simultaneously on a data item.

Exclusive lock:

- These Locks are referred as Write locks, and denoted by **'X'.**
- If a transaction T has obtained Exclusive lock on data item X, then **T can be read as well as write X**.
- Only one Exclusive lock can be placed on a data item at a time.
- This means multiple transactions does not modify the same data simultaneously.

Shared / Exclusive Rules:

1. A transaction T must issue the operation ***read_lock(X)*** or ***write_lock(X)*** before any ***read_item(X)*** operation is performed in T.
2. A transaction T must issue the operation write_lock(X) before any ***write_item(X)*** operation is performed in T.
3. Transaction T must issue the operation unlock(X) after all ***read_item(X)*** and ***write_item(X)*** operations are completed in T.
4. A transaction will not issue a ***read_lock(X)*** operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
5. A transaction will not issue a ***write_lock(X)*** operation if it already holds a read (shared) lock or a write (exclusive) lock on item X.
6. A transaction will not issue a ***unlock(X)*** operation unless it already holds a ***read_lock(shared)*** or a ***write_lock (exclusive)*** on item X.

Example:

T1:	**Lock-X(B)**
	Read(B)
	B:=B-50
	Write(B)
	Unlock(B)
	Lock-X(A)
	Read(A)
	A:=A+50
	Write(A)
	Unlock(A)

T2:	**Lock-S(A)**
	Read(A)
	Unlock(A)
	Lock-S(B)
	Read(B)
	Unlock(B)
	Display(A+B)

3) **TWO-PHASE LOCKING (2PL):**

- Two-phase locking (**2PL**) is a method of controlling concurrent processing in which all locking operations precede the first unlocking operation.

- Thus, a transaction is said to follow the two-phase locking protocol if all locking operations (read_Lock, write_Lock) precede the first unlock operation in the transaction.
- Two-phase locking is the standard protocol used to defines how transactions acquire and hand over locks.
- This protocol requires that each transaction issue lock and unlock requests in two phases.
  1. **Growing Phase**: In this phase, a transaction may obtain locks, but may not release any lock.
  2. **Shrinking Phase**: In this phase, a transaction may release locks, but may not obtain any new locks.
- Initially, a transaction is in the ***growing phase***. The transaction acquires locks as needed.
- Once the transaction releases a lock, it enters in the ***shrinking phase***, and it cannot issue more lock request.

Example: 01

T3	**Lock-X(B)**
	Read(B)
	B:=B-50
	Write(B)
	Lock-X(A)
	Read(A)
	A:=A+50
	Write(A)
	Unlock(B)
	Unlock(A)

Example: 02

T1	T2
Raed_lock(Y)	Read_lock(X)
Read_item(Y)	Read_item(X)
Unlock(Y)	**Unlock(X)**
Write_lock(X)	**Write_lock(Y)**
Read_item(X)	Read_item(Y)
X:X+Y	Y:=X+Y
Write_item(X)	Write_item(Y)
Unlock(X)	Unlock(Y)

Not obey two phase locking

Example: 03

T1	T2
Raed_lock(Y)	Read_lock(X)
Read_item(Y)	Read_item(X)
Write_lock(X)	**Write_lock(Y)**
Unlock(Y)	**Unlock(X)**
Read_item(X)	Read_item(Y)
X:X+Y	Y:=X+Y
Write_item(X)	Write_item(Y)
Unlock(X)	Unlock(Y)

Obey two phase locking

Advantages:

1. The two-phase locking ensures conflict serializability.

Disadvantages:

1. Two-phase locking does not ensure freedom from deadlock.
2. Cascading Roll-backing may occur under two-phase locking.

STRICT TWO-PHASE LOCKING (2PL):

- Strict two-phase locking method used in concurrent systems.
- Strict Two-Phase is most used in real-world implementation.

- **Rules of Strict 2PL**:

1. If a transaction T wants to read/write an object, it must request a shared/exclusive lock on the object.
2. All exclusive locks held by transaction T are released when T commits.
3. A Transaction T does not release any of its Exclusive Lock until it commits or Aborts.
4. Hence, no other transaction can read or write an item that is written by T unless T has committed.

T1	T2
Lock-S(A)	
Read(A)	
	Lock-S(A)
	Read(A)
	Loc-X(B)
	Read(B)
	Write(B)
	Commit
Lock-X(C)	
Read(C)	
Write(C)	
Commit	

- Strict Two-phase locking prevents transaction from reading uncommitted data, overwriting uncommitted data and unrepeatable reads.
- Strict 2PL prevents Cascading Rollbacks.
- Strict 2PL does not guarantee a Deadlock Free Schedule.

RIGOROUS TWO-PHASE LOCKING (2PL):

- This protocol that all locks be held until the transactions commits.

C. **DEADLOCKS :**

Definition:

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

"There exists a set of transactions {T_0, T_1, T_2,….T_n} such that T_0 is waiting for a data item that T_2 holds and T_2 waiting for a data item that T_3 holds…. T_n is waiting for data item that T_0 holds. In such situation, none of the transaction can make progress"

T1	T2
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-X(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

- Here T1 is holding an Exclusive lock on B and t2 is requesting a Shared lock on B, T2 is waiting for T1 to unlock B.
- Similarly, T2 is holding Shared lock on A, T1 is requesting an Exclusive lock on A, T1 is waiting for T2 to unlock A.
- In this situation neither of transaction can proceed with normal execution.
- This situation is called DEADLOCK.

There are two principle methods for dealing with deadlock problem

1. **Deadlock Prevention**: This approach ensures that the system will never enter in deadlock state.
2. **Deadlock Detection and Recovery**: This approach tries to recover from Deadlock if system enters in deadlock state.

Deadlock Prevention:

- Deadlock prevention technique avoids the conditions that lead to deadlocking.
- It requires that every transaction lock all data items it needs in advance.
- In other words, a transaction requesting a new lock is aborted if there is the possibility that a deadlock can occur.

There are two approaches for deadlock prevention:

Approach: 1

- This ensures that no cyclic wait can occur by ordering the request for locks.
- It requires that every transaction lock all data items it needs in advance.
- It is required that, either **all data items should be locked** in one step, or **none should be locked.**

Disadvantages of this approach-1:

1. It is **hard to predict** before the transaction begins, What data items need to be locked.
2. Data item **utilization may be very low**, because many of the data items may be locked but unused for a long time.

Approach: 2

- Deadlock prevention is to use **preemption** and **transaction rollbacks**.
- In preemption when a transaction T2 request a lock that transaction T1 holds.
- The lock granted to T1 may be preempted by rolling back T1, and granting of lock to T2.
- To control preemption, a unique timestamp is assigned to each transaction.
- The system uses timestamp to decide whether a transaction wait or roll back.
- Two different deadlock prevention schemes using timestamp are:
  - Wait Die.
  - Wound Wait.

1. **Wait Die**:

- The Wait-Die scheme is non-preemption technique.
- When Transaction T_i request a data item held by T_j, T_i is allowed to wait only if it has a timestamp smaller than T_j.
- Otherwise, T_i is rolled back (Die)

Example:

- Consider three transactions T1, T2, T3 with timestamps 5, 10, 15 respectively.
- If T1 request a data item held by T2, Then T1 will wait.
- If T3 request data item held by T2, then T2 will be Rolled Back.

2. **Wound Wait**:

- The Wound Wait scheme is preemption technique.
- When Transaction T_i request a data item held by T_j, T_i is allowed to wait only if it has a timestamp greater than T_j.
- Otherwise, T_i is rolled back (Die)

Example:

- Consider three transactions T1, T2, T3 with timestamps 5, 10, 15 respectively.
- If T1 request a data item held by T2, Then T2 will be Rolled Back.
- If T3 request data item held by T2, then T3 will wait.

Deadlock detection:

- This technique allows deadlock to occur, but then, it detects it and solves it.
- Here, a database is periodically checked for deadlocks.
- If a deadlock is detected, one of the transactions, involved in deadlock cycle, is aborted. other transaction continue their execution.

An aborted transaction is rolled back and restarted.

4. **DATABASE RECOVERY TECHNIQUES**

Database Failure:

- We cannot guarantee a failure-free transaction every time. There are several types of failures that affect database processing.
- Some failures may affect the disk storage, while some may only affect the data items of the database residing in the main memory.

Types Failure

- Failures are generally classified as Transaction, System, and Media Failures.
- There are several possible reasons for a transaction to fail in the middle of execution.

1. **A Computer Failure** (System Crash):
   - A hardware, software, or network error occurs in the computer system during transaction execution.
   - Hardware crashes are usually Media Failure

- Example: Main Memory Failure.

2. **A Transaction or System error**:
   - Some operation in the transaction may cause it to fail, such as **integer overflow** or **divide by zero**.
   - Transaction Failure may also occur because of **erroneous parameter values** or because of a **logical programming error**.

3. **Disk Failure**:
   - Some disk blocks may lose their data because of a read or write malfunction or because of a read/write head crash.
   - This may happen during a read or a write operation of the transaction.

4. **Physical Problem and Catastrophes**:
   - This refers to an endless list of problems that includes power failure or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

- Failure of types 1 and 2 are more common.
- Whenever a failure of type 1 and 2 occurs, the system must keep sufficient information to recover from the failure.
- Disk Failure and Catastrophic Failure are do not happen frequently.
- If they occur, recovery is a major task.

Database Recovery Techniques

- Recovery from transaction failure means that the database is restored to the most recent consistent state just before the time of failure.
- Wide portion of the database due to catastrophic failure, the recovery method **restores past copy** of the database and **reconstructs** a more current state by **redoing** the operations **up to the time of failure**.
- When the database is not physically damaged, the recovery system is to reverse any changes that caused the inconsistency by undoing some operations.

- It may also redo some operation in order to restore a consistent state of the database.

There are many different approaches to recover a database: Manual reprocessing, a variety of automated recovery techniques.

1. Manual Reprocessing

- The database is periodically backed up and all transactions applied since the last save are recorded.
- If the system crashes, the latest database save is restored and all of the transactions are re-applied (by users) to bring the database back up to the point just before the crash.

Limitations: 1) **Time required** reapplying transactions.
2) Transactions **might have other** (physical) potential **failures**.
3) Reapplying concurrent transactions is **not straight forward**.

2. **Automated Recovery:**

There are several types of automated recovery techniques including: deferred update and immediate update, shadow paging, etc.

A transaction can be in one of the following states:

1. **Active** - When the transaction just begins
2. **Partially Committed** - After the last operation has completed (but before the commit point is reached)
3. **Failed** - Normal operation is prevented.
4. **Aborted** - Transaction is rolled back.
5. **Committed** - Transaction completes all operations and moved the database to the next consistent state.

Each transaction writes the following information to the log:

6. Start(T) - the fact that transaction T has started.
7. Write(T, X, old_value, new_value) - The Transaction T has written to item X with the new_value. old_value is also maintained.
8. Read(T, X) - The Transaction T has read data item X
9. Either
10. Commit(T) - Transaction T committed, or Abort(T) - transaction T was aborted

Checkpoint:

- A recovery point in the logs where all current transactions have terminated and all updates have been written to disk. *Consists of 4 steps*:
  1. Cease accepting new transactions
  2. Allow all unfinished transactions to complete (commit or abort)
  3. Write all pending changes to disk and to logs
  4. Resume accepting new transactions
- In many environments, it is possible to take checkpoints each 15 minutes or half hour, etc.
- Recovery must then only be done from the time of the last checkpoint.

1. DEFERRED UPDATE RECOVERY:

- Also called NO-UNDO/REDO
- During a transaction, only record the changes to data items in the log.
- When the transaction commits, update the data items on disk.
- Two main rules:
  1. A transaction cannot change any items in the database until it commits.
  2. A transaction may not commit until the entire write operations are successfully recorded in the log.

     This means that we must check the log is actually written to disk.

```
T1:    Ra Rd Wd C
T2:    Rb Wb Rd Wd C
T3:    Ra Wa Rc Wc C
T4:    Rb Wb Ra Wa C

Log file:
Start(T1)
Write(T1, d, old, new)
Commit(T1)
checkpoint
Start(T4)
Write(T4, b, old, new)
Write(T4, a, old. new)
Commit(T4)
Start(T2)
Write(T2, b, old, new)
Start(T3)
Write(T3, a, old, new)
    s y s t e m  c r a s h
```

- Since T1 and T4 committed, their changes were written to disk.
- However, T2 and T3 did not commit, hence their changes were not written to disk.
- T2 and T3 are ignored because they did not reach their commit points.
- T4 is redone because its commit point is after the last system checkpoint.

Advantages:

1. Recovery is made easier:

   Any transaction that reached the commit point (from the log) has its writes applied to the database. All other transactions are ignored.

2. Cascading rollback does not occur because no other transaction sees the work of another until it is committed.

Disadvantages:

1. Concurrency is limited: Must employ Strict 2PL which limits concurrency.

2. **IMMEDIATE UPDATE RECOVERY**

- ❖ A Transaction issues an update command; the database can be updated immediately, without any need to wait for the transaction to reach its commit point.
- ❖ An update operation must still be *recorded in the log (on disk*) before it is applied to the database – using the ***write ahead logging protocol***.
- ❖ So that we can recover in case of failure.
- ❖ Immediate Update allows the write operations to the database as the transaction is executing.
- ❖ Writes are still saved in the log before being applied to the database - a ***Write-Ahead Log*** (WAL)
- ❖ Maintain two logs:
- ❖ **REDO log**: A record of each new data item in the database.

  UNDO log: A record of each updated data item (old values).

Two rules:

1. Transaction T may not update the database until all UNDO entries have been written to the UNDO log.
2. Transaction T is not allowed to commit until all REDO and UNDO log entries are written (*forced-written* to disk).

To Recover:

1. Begin at the end of the log and read backwards to the last checkpoint Create two lists:
   - ✓ **C** - transactions that have committed (Committed Transaction)
   - ✓ **NC** - transactions that did not commit (Active Transaction)

- ✓ Undo all the write operations of the Active transactions from the log, using UNDO procedure.
- ✓ Redo the write operations of the Committed Transactions from the log using REDO procedure.

```
T1:    Ra Rd Wd C
T2:    Rb Wb Rd Wd C
T3:    Ra Wa Rc Wc C
T4:    Rb Wb Ra Wa C

Log file:
Start(T1)
Write(T1, d, old, new)
Commit(T1)
checkpoint
Start(T4)
Write(T4, b, old, new)
Write(T4, a, old. new)
Commit(T4)
Start(T2)
Write(T2, b, old, new)
Start(T3)
Write(T3, a, old, new)
    s y s t e m  c r a s h
```

- ❖ Since T1, T2, T3 and T4 changes were written to disk.
- ❖ However, T1 and T4 are committed.
- ❖ T2 and T3 are not committed.
- ❖ T2 and T3 are in Active Transaction List and T4 is in Committed Transaction.
- ❖ T2 and T3 are Undone from the log using UNDO procedure.
- ❖ T4 is redone because its commit point is after the last system checkpoint using REDO procedure.

Advantages:

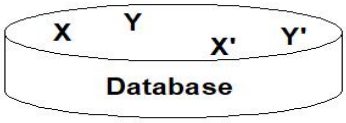
Immediate update **allows higher concurrency** because transactions write continuously to the database rather than waiting until the commit point.

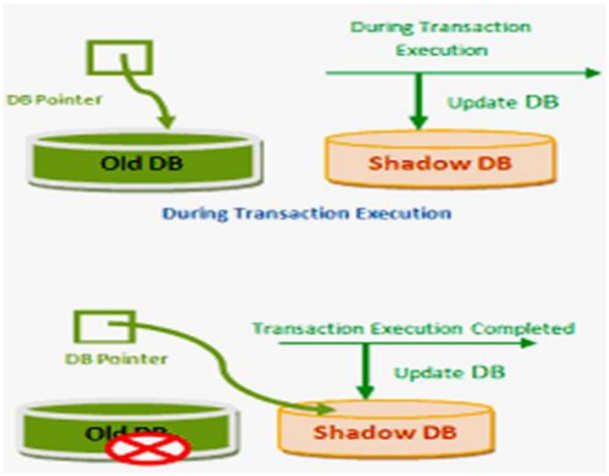
Disadvantages:

Can lead to cascading rollbacks - time consuming and may be problematic.

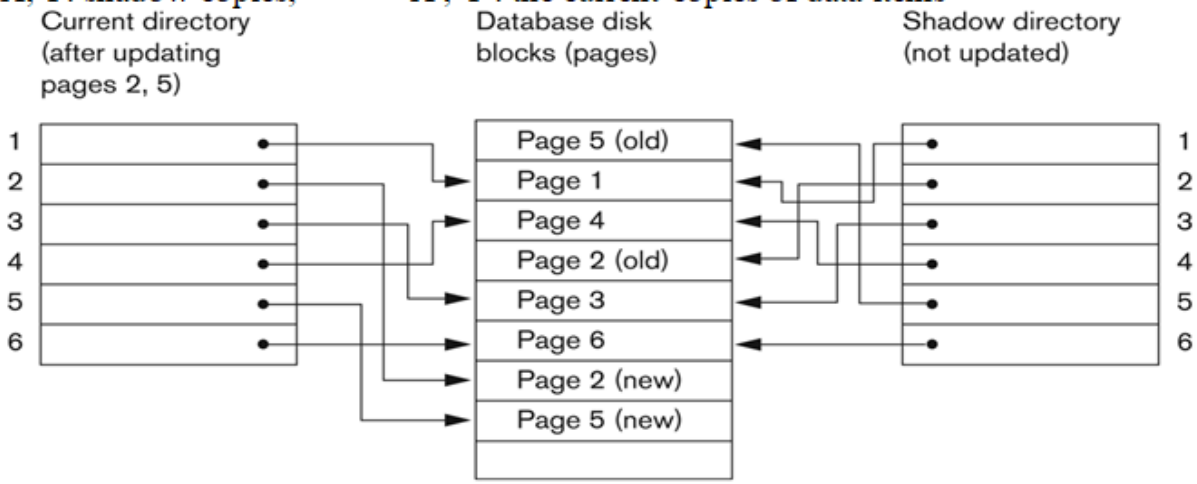
3. **SHADOW PAGING:**

- ❖ The AFIM (After Image) does not overwrite its BFIM (Before Image) but recorded at another place on the disk.
- ❖ Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.





X, Y: shadow copies; X', Y': the current copies of data items



- To recover from a failure, it is sufficient to free the modified pages and discard the current directory.
- The state of the database before transaction execution is available through the shadow directory.
- Database can be returned to its previous state that was executing when the crash occurred.
- We can categorize as a **NO-UNDO/NO-REDO** technique for recovery Logs and checkpoints must be incorporated into the shadow paging technique.
- **Disadvantages:**

Complex storage management strategies, the overhead of writing shadow directories to disk, garbage collection overhead (old pages referenced by the shadow directory).

Unit – 5

Database Security and Object – Based Databases

Unit – 5

Database Security and Object – Based Databases

Syllabus:

Database Security-Authentication, Authorization and access control - DAC, MAC and RBAC models –SQL Injection-Object based databases- Object oriented and object relational databases- Persistent Programming Languages-Object-Relational Mapping-Object-Oriented versus Object-Relational Databases – Logical and web databases-Overview of Data warehousing - Data mining techniques.

Security Risk to Database Includes

- Unauthorized database users.
- Unauthorized Database Administrator
- Unauthorized access to Database
- Unauthorized alternation to available data
- Lack of access to Database services

Sensitive data includes

- ✓ Bank/Demat accounts
- ✓ Credit card, Salary,
- ✓ Income tax data\
- ✓ University admissions, marks/grades
- ✓ Land records, licenses

Database Security

Database Security is defined as the process by which "Confidentiality, Integrity and Availability" of the database can be protected.

Database Security Concepts

1. Confidentiality
2. Integrity
3. Availability

Database Security Concepts

(1) Confidentiality

- Enforced by encrypting the data in the stored database.
- Encryption is a technique or a process by which the data is encoded in such a way that only that authorized users are able to read the data.
- Encryption is rendering sensitive data unreadable to unauthorized users.

Database Security Concepts

(2)Integrity

- Enforced by defining which user has to be given permission to access the data in the database.
- **Example:** Data related to employee may have permission for viewing records and altering only the part of information like his contact details, where as the person like Human resource manager will have more privileges.

Database Security Concepts

(3)Availability

- Database must have not unplanned downtime.
- To ensure this ,following steps should be taken –

1. Restrict the amount of the storage space given to each user in the database.
2. Limit the number of concurrent sessions made available to each database user.
3. Back up the data at periodic intervals to ensure data recovery in case of application users.

Threats to Database

- ✓ SQL Injection.
- ✓ Unauthorized access.
- ✓ Password Cracking.
- ✓ Network EavesDropping.

Threats to Database

(1)SQL Injection

A form of attack on a database-driven Web site in which the attacker executes unauthorized SQL commands by taking advantage of insecure code on a system connected to the Internet, bypassing the firewall.

Vulnerabilities:

- ✓ Poor Input validation to web application.
- ✓ Unsafe ,dynamically constructed SQL commands.
- ✓ Weak permissions that fail to restrict the application to Database

Threats to Database

(1)SQL Injection

Countermeasures:

- ✓ Your application should constrain and sanitize input data before using it in SQL queries.
- ✓ Use type safe SQL parameters for data access. These can be used with stored procedures or dynamically constructed SQL command strings. Using SQL parameters ensures that input data is subject to type and length checks.
- ✓ Use a SQL Server login that has restricted permissions in the database. Ideally, you should grant execute permissions only to selected stored procedures in the database and provide no direct table access.

Threats to Database

(2)Unauthorized Access

Direct access to your database server should be restricted to specific client computers to prevent unauthorized server access.

Vulnerabilities:

- ✓ Failure to block the SQL Server port at the perimeter firewall
- ✓ Lack of IPSec or TCP/IP filtering policies

Threats to Database

(2)Unauthorized Access

Countermeasures:

- ✓ Make sure that SQL Server ports are not visible from outside of the perimeter network.
- ✓ Within the perimeter, restrict direct access by unauthorized hosts, for example, by using IPSec or TCP/IP filters.

Threats to Database

(3) Password cracking

- ✓ A common first line of attack is to try to crack the passwords of well known account names, such as SA (the SQL Server administrator account).

Vulnerabilities:

- ✓ Weak or blank passwords
- ✓ Passwords that contain everyday words

Countermeasures:

- ✓ Create passwords for SQL Server login accounts that meet complexity requirements.
- ✓ Avoid passwords that contain common words found in the dictionary.

Threats to Database

(4) Network Eavesdropping

- ✓ Eavesdropping refers to unauthorized access of reading messages.
- ✓ The deployment architecture of most applications includes a physical separation of the data access code from the database server.
- ✓ As a result, sensitive data, such as application-specific data or database login credentials, must be protected from network eavesdroppers.

Threats to Database

(4) Network Eavesdropping

Vulnerabilities:

- ✓ Insecure communication channels
- ✓ Passing credentials in clear text to the database; for example:
  - Using SQL authentication instead of Windows authentication
  - Using SQL authentication without a server certificate

Methods of securing the database

- ✓ Authorization - privileges, views.
- ✓ Authentication – passwords.
- ✓ Encryption - public key / private key, secure sockets.
- ✓ Logical - firewalls, net proxies.

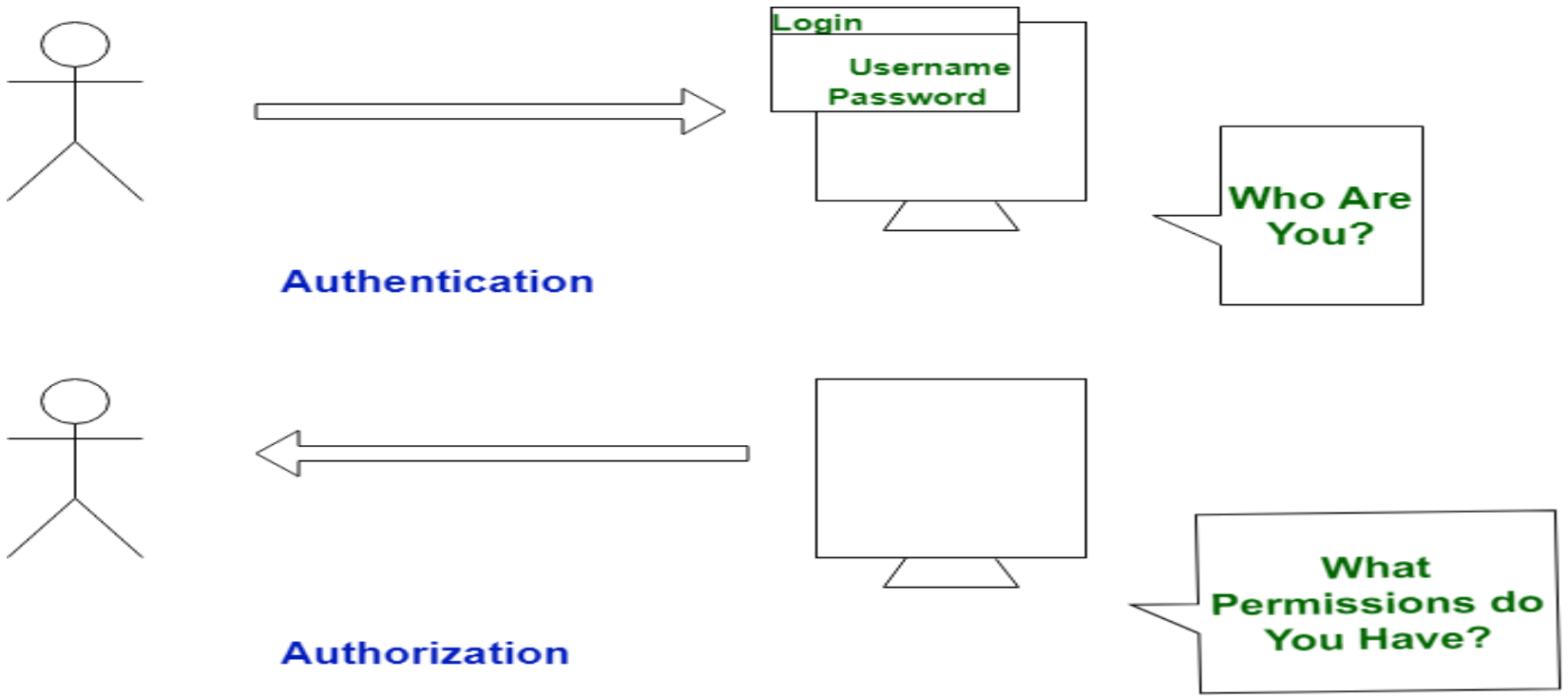
(1) Authorization

- Authorization is a process by which a server determines if the client has permission to use a resource or access a file.
- Authorization is usually coupled with authentication so that the server has some concept of who the client is that is requesting access.
- The type of authentication required for authorization may vary; passwords may be required in some cases but not in others.
- In some cases, there is no authorization; any user may be use a resource or access a file simply by asking for it.

(2) Authentication

- Authentication is used by a server when the server needs to know exactly who is accessing their information or site.
- In Authentication, the user or computer has to prove its identity to the server or client.
- Usually, Authentication by a server entails the use of a user name and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.
- Authentication by a client usually involves the server giving a certificate to the client in which a trusted third party.

Authentication Vs Authorization



Authentication Vs Authorization

S.NO	AUTHENTICATION	AUTHORIZATION
1.	In authentication process, the **identity of users are checked** for providing the access to the system.	While in authorization process**, person's or user's authorities are checked** for accessing the resources.
2.	In authentication process, users or persons are **verified.**	While in this process, users or persons are **validated.**
3.	It is **done before the authorization** process.	While this process is **done after the authentication** process.
4.	It needs usually **user's login details**.	While it needs **user's privilege or security levels**.

(3) Security of the database Through Abstraction

- Data encryption enables to encrypt sensitive data, such as credit card numbers, stored in table columns.
- Encrypted data is decrypted for a database user who has access to the data.
- Data encryption helps protect data stored on media in the event that the storage media or data file gets stolen.

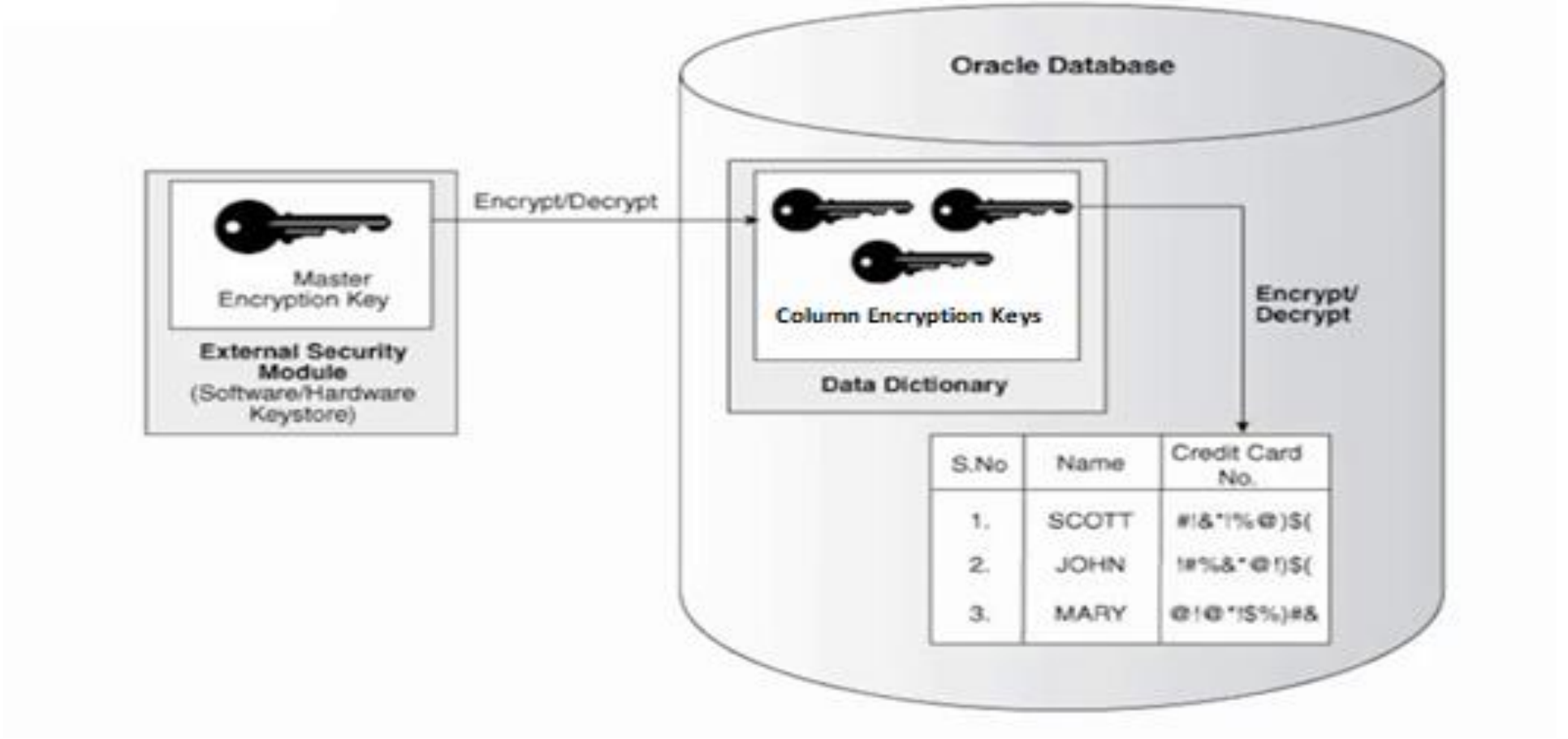
(3) Security of the database Through Abstraction

Encryption:

- Data encryption is a key-based access control system. Even if the encrypted data is retrieved, it cannot be understood until authorized decryption occurs.
- When a table contains encrypted columns, a single key is used regardless of the number of encrypted columns. This key is called the column encryption key.
- The column encryption keys for all tables, containing encrypted columns, are encrypted with the database server master encryption key and stored in a dictionary table in the database.
- The master encryption key is stored in an external security module that is outside the database and accessible only to the security administrator.

(3) Security of the database Through Abstraction

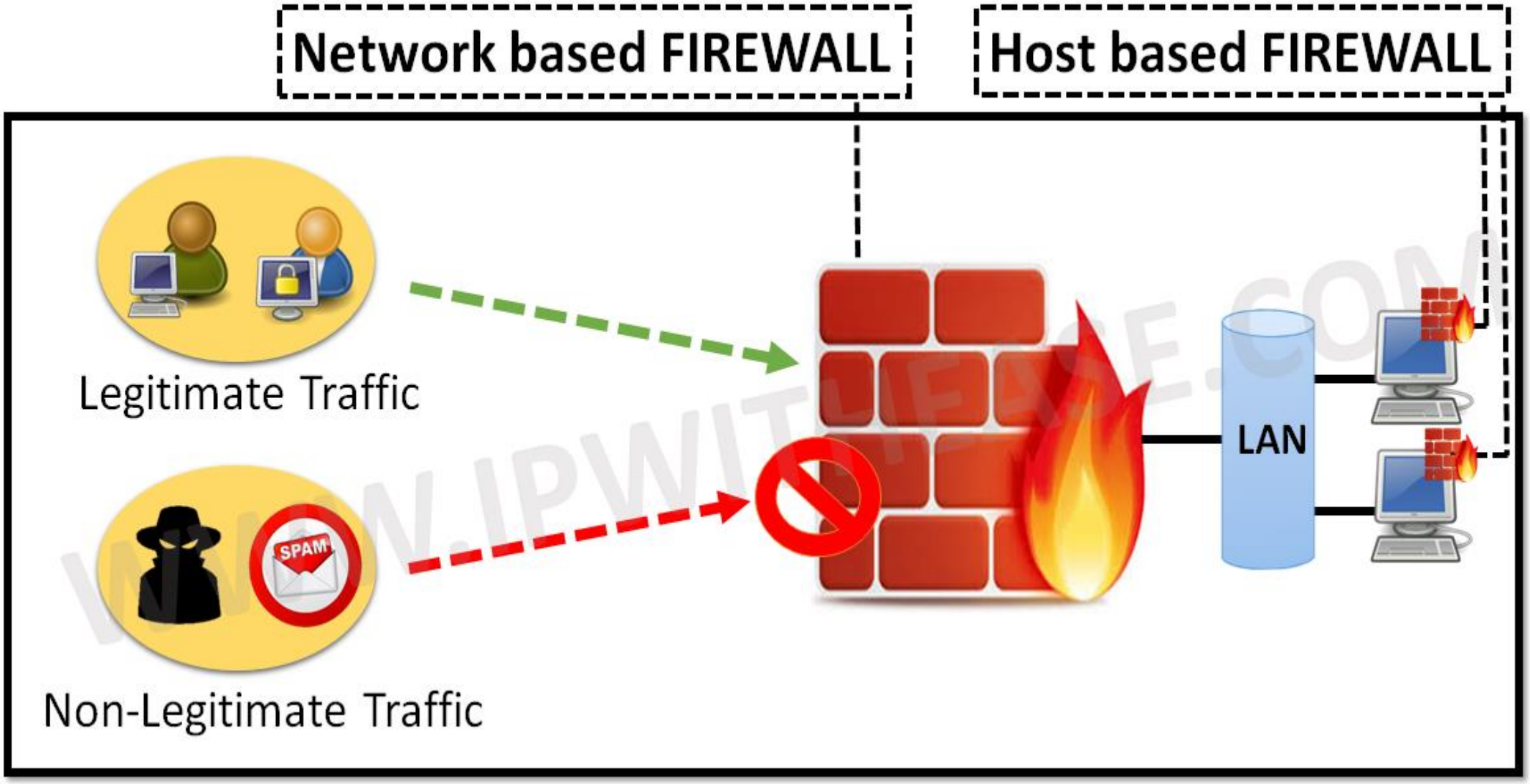
Encryp



(4) Security of the database through FIREWALLS

- A FIREWALL is dedicated software on another computer which inspects network traffic passing through it.
- It denies (or) permits passage based on set of rules.
- Database Firewalls are a type of Web Application Firewalls that monitor databases to identify and protect against database specific attacks that mostly seek to access sensitive information stored in the databases.

(4) Security of the database through FIREWALLS



ACCESS CONTROL

ACCESS CONTROL

What is Access Control in Database Security?

- Database access control is a method of allowing access to company's sensitive data only to those people (database users) who are allowed to access such data and to restrict access to unauthorized persons.
- It includes two main components: Authentication and Authorization.
- Note that Authentication & Authorization are not enough to protect data.
- An additional layer of security is required.
- Without authentication and authorization, there is no data security.

TYPES OF ACCESS CONTROL

Access Control Models include –

1. Discretionary Access Control (DAC)
2. Mandatory Access Control (MAC).
3. Role Based Access Control (RBAC) - is the most common method today.
4. Attribute Based Access Control (ABAC).

TYPES OF ACCESS CONTROL

(1) Discretionary Access Control (DAC):

- DAC models, the Data Owner Allows Access.
- DAC is a means of assigning access rights based on user-specified rules.
- DAC is the least restrictive compared to the other systems

TYPES OF ACCESS CONTROL

(2) Mandatory Access Control (MAC)

- MAC was developed using a nondiscretionary model, in which people are granted access based on an information clearance.
- MAC is a policy in which access rights are assigned based on central authority regulations.

TYPES OF ACCESS CONTROL

(3) Role Based Access Control (RBAC)

- RBAC grants access based on a user's role and implements key security principles such as "least privilege" and "separation of privilege".
- Thus, someone attempting to access information can only access data necessary for their role.

TYPES OF ACCESS CONTROL

(4) Attribute Based Access Control (ABAC)

- In ABAC, each resource and user are assigned a series of attributes.
- In this dynamic method, a comparative assessment of the user's attributes, including time of day, position and location, are used to make a decision on access to a resource.

SQL INJECTION

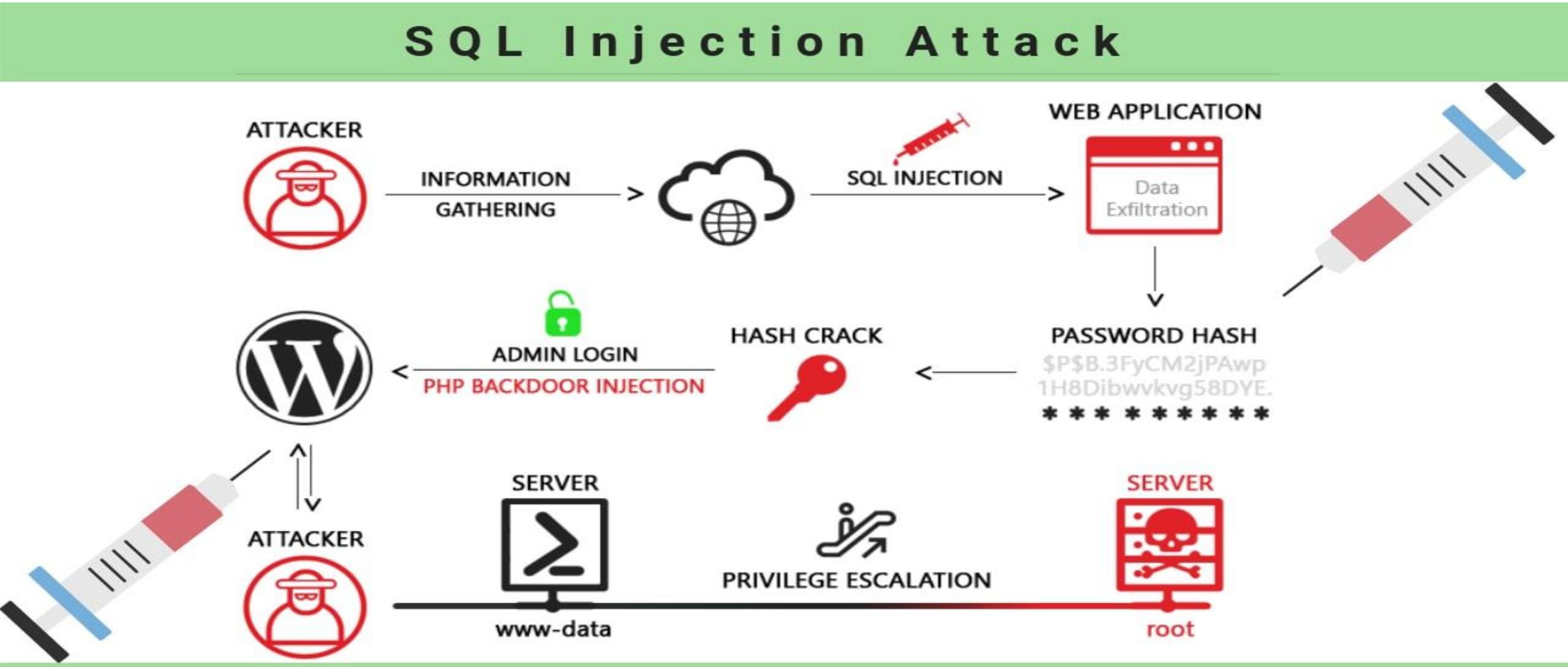
SQL INJECTION

- SQL Injection (SQLi) is a type of an attack that makes it possible to execute malicious SQL statements.
- These SQL statements control a database server behind a web application.
- Attackers can use SQL Injection vulnerabilities to bypass application security measures.
- They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database.
- They can also use SQL Injection to add, modify, and delete records in the database.

SQL INJECTION – ATTACK INTENSION

1. Determining database schema
2. Extracting data
3. Adding or modifying data
4. Bypassing authentication

SQL INJECTION - INTENSION



HOW SQL INJECTION ATTACK WORKS?

1. The types of attacks that can be performed using SQL injection vary depending on the type of database engine.
2. The attack works on dynamic SQL statements.
3. A dynamic statement is a statement that is generated at run time using parameters password from a web form or URI query string.

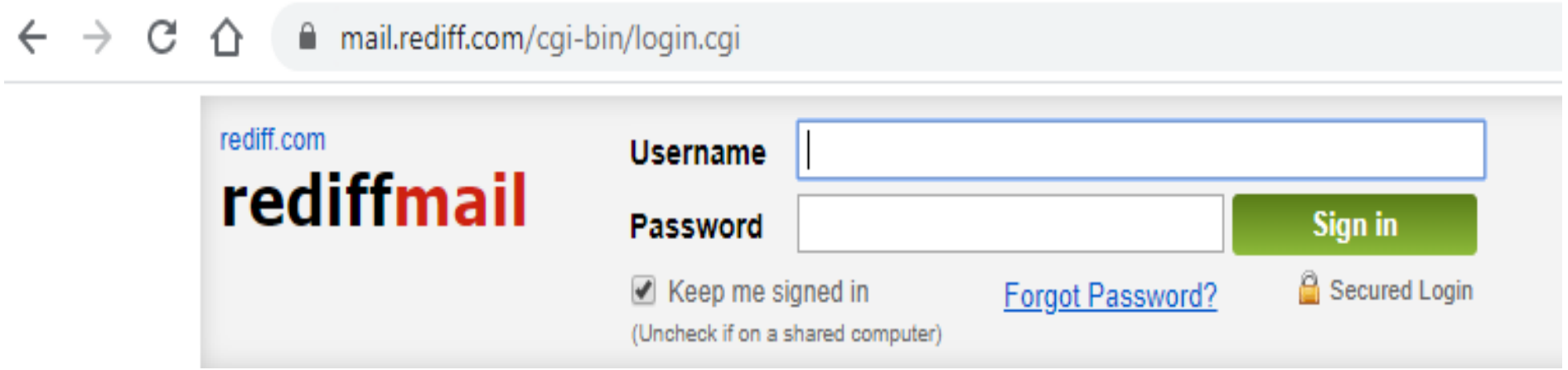
HOW SQL INJECTION ATTACK WORKS?

Simple Web Application with LOGIN FORM:

```
<form action='index.php' method="post">

<input type="email" name="email" required="required"/>

<input type="password" name="password"/>

 <input type="checkbox" name="remember_me" value="Remember me"/>

<input type="submit" value="Submit"/> </form>
```

HOW SQL INJECTION ATTACK WORKS?

Simple Web Application with LOGIN FORM:



HOW SQL INJECTION ATTACK WORKS?

The statement at the backend for checking user ID is as follows

```
SELECT * FROM users WHERE email = $_POST['email'] AND password = md5($_POST['password']);
```

HERE,

1. The above statement uses the values of the $_POST[] array directly without sanitizing them.
2. The password is encrypted using MD5 algorithm.

HOW SQL INJECTION ATTACK WORKS?

```
CREATE TABLE `users` (
 `id` INT NOT NULL AUTO_INCREMENT,
 `email` VARCHAR(45) NULL,
 `password` VARCHAR(45) NULL,
 PRIMARY KEY (`id`));

insert into users (email,password) values ('m@m.com',md5('abc'));
```

STEP 1

STEP 2

Build Schema | Edit Fullscreen | Browser | [;]

```
select * from users
```

STEP 3

STEP 4

Run SQL | Edit Fullscreen | Format Code | [;]

ID	EMAIL	PASSWORD
1	m@m.com	900150983cd24fb0d6963f7d28e17f72

HOW SQL INJECTION ATTACK WORKS?

Suppose user supplies admin@admin.sys and 1234 as the password. The statement to be executed against the database would be

```
SELECT * FROM users WHERE email = 'admin@admin.sys' AND password = md5('1234');
```

An Attacker generated dynamic statement will be as follows.

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1 = 1 LIMIT 1 -- ' ] AND password = md5('1234');
```

HERE,

- xxx@xxx.xxx ends with a single quote which completes the string quote.
- **"OR 1 = 1 LIMIT 1"** is a condition that will always be true and limits the returned results to only one record.
- **"-- ' AND"** … is a SQL comment that eliminates the password part.

HOW SQL INJECTION ATTACK WORKS?

TYPES OF SQL INJECTION ATTACK

1. In-band SQLi
    a) Error-based SQLi
    b) Union-based SQLi
2. Blind SQLi
    a) Boolean
    b) Time-based
3. Out-of-band SQLi

How to Prevent SQL Injections

Step 1: Train and maintain awareness

Step 2: Don't trust any user input

Step 3: Use whitelists, not blacklists

Step 4: Adopt the latest technologies

Step 5: Employ verified mechanisms

Step 6: Scan regularly

OODB
Object Oriented Databases

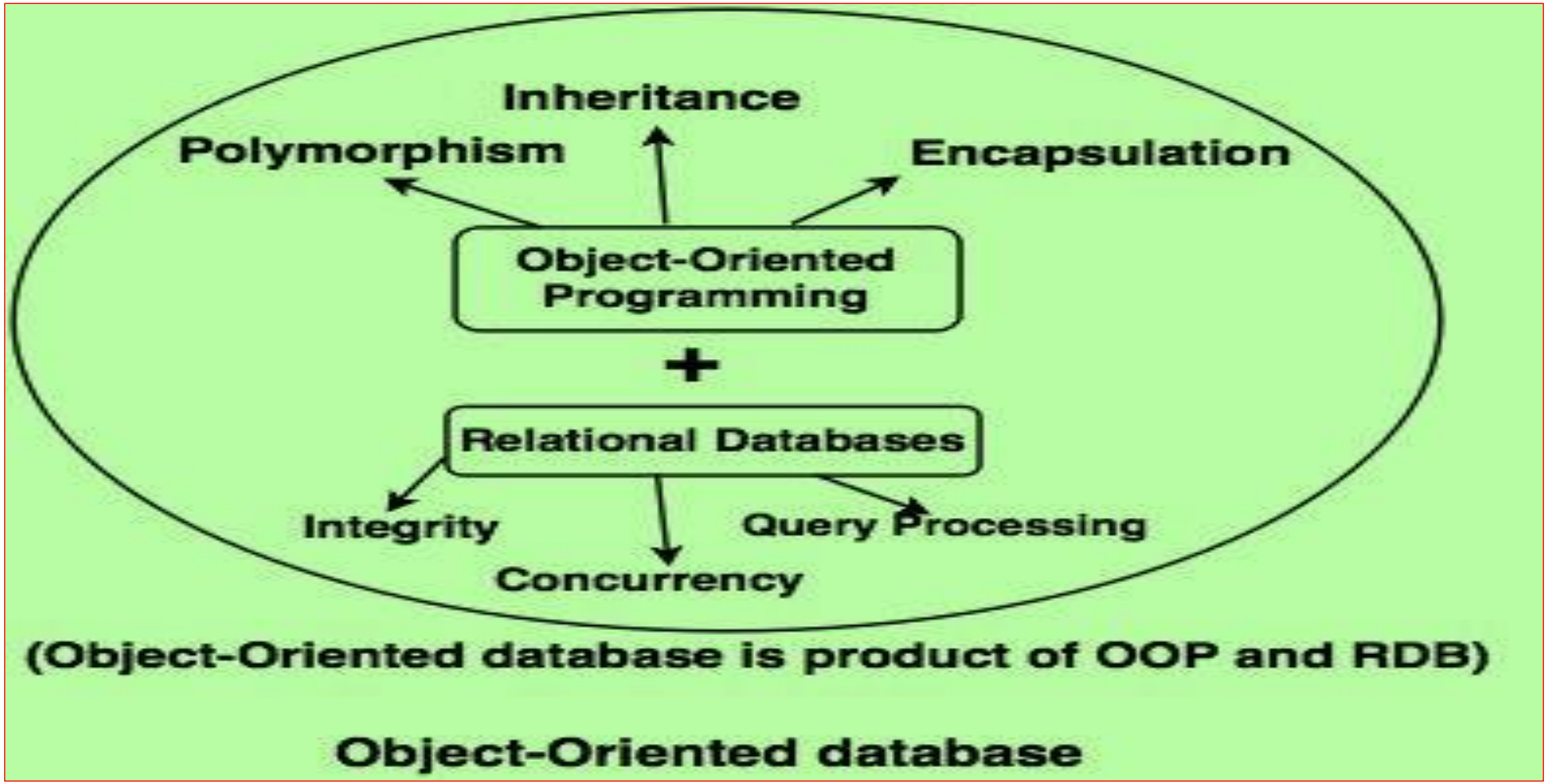
DATA MODELS

- Hierarchical model
- Network Model
- Relational Model
- Object Model

OODB

- Object Oriented Database Systems are alternative to Relational Database.
- In Object Oriented Database, information is represented in the form of Objects.
- If we can combine the features of Relational Model and Object Oriented Databases, the resultant model is called as Object Oriented Database Model.

OODB


Inheritance
Polymorphism
Encapsulation
Object-Oriented Programming
+
Relational Databases
Integrity
Query Processing
Concurrency
(Object-Oriented database is product of OOP and RDB)
Object-Oriented database

OODB

- An Object-Oriented Database System should satisfy two criteria:
  - ✓ It should be a DBMS, and it should be an Object-Oriented System.
- OODB implements OO concepts such as Object Identity, Polymorphism, Encapsulation and Inheritance.
- Object Oriented Databases are designed to work well with Object Oriented Programming languages such as Python, Java, Objective-C.

OODB Architecture

OODB architecture can be divided into two categories:

1. **Standalone OODB** where all of the data available is stored in object data model. Thus there is no overhead in mapping objects to application objects.
2. **Object Relational DBMS:** OODB acts as a staging layer for existing data in relational database. The data in relational database are mapped to object models and stored in object data database.

Object Definition Language (ODL)

Object Definition Language (ODL)

- ODL is the standardized language for defining the structure of database with respect to the object data model.
- ODL creates a layer of abstraction making data language and database independent to allow applications to move between databases or different language implementations.
- ODL defines three components of the object oriented data model:

1. Abstraction
2. Inheritance
3. Encapsulation

Object Definition Language (ODL)

(1) Data Abstraction

- In OODB, Abstract Data Model is implemented as a GRAPH.
  - ✓ Vertices representing the Objects
  - ✓ Edges representing Relations.
- **Object Instance**: An Entity in an object model is called an Object Instance.
- **Object Identification:** Every Object Instance has a Unique Identity. This id used to reference object instances.

Object Definition Language (ODL)

(1) Data Abstraction

- **Object Class:** Similar Object Instances are grouped together into a class.
- **Object Reference Or Relationships:** The Object Model directly supports references. Object Instances (Entity) refer other using Object Identities.

Object Definition Language (ODL)

(2) Data Encapsulation:

- Encapsulation in the Object Model allows the Object Instances defined by the class.
- This allows Code and Data to be Packaged Together.
- This includes an Interface and an Implementation for each object.

Object Definition Language (ODL)

(3) Data Inheritance:

- Derived directly from Object Oriented Programming Languages.
- Object Data Models also allow for Inheritance, Polymorphism and Overriding.

Object Query Language (OQL)

- Object Query Language allows SQL-like queries to be performed on a OODB.
- Like-SQL, it is a Declarative Language.
- Query Structures look very similar in SQL and OQL but the results returned are different.

Object Query Language (OQL)

- Example: OQL query to obtain Voter Names who are from the state of kerala.
  - Select distinct Voter_Name from voters where State = “AP”

Voter_ID	Voter_Name	State
445620551	Siva Sai Kiran	TS
415320529	Mani Deep	AP
545620515	Abhiram N	AP
414420531	Ajay Ram	TN

Result from SQL Query

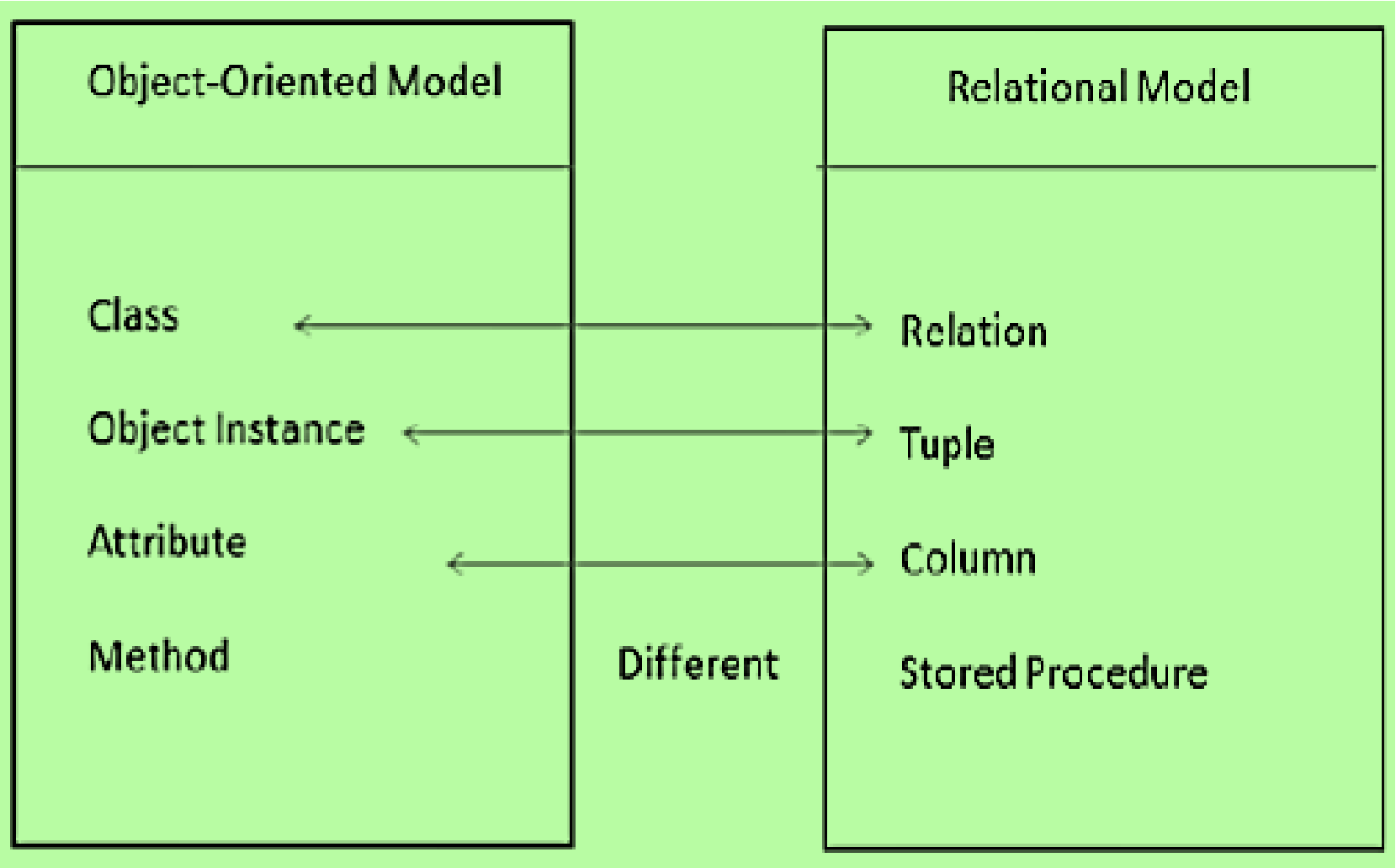
Voter_Name
Mani Deep
Abhiram N

Result from OQL Query

String
Mani Deep

String
Abhiram N

Object Vs Relational Model



Object Vs Relational Model

Key	RDBMS	OODBMS
Definition	RDBMS stands for Relational Database Management System.	OODBMS stands for Object Oriented Database Management System.
Data Management	Data is stored as entities defined in tabular format.	Data is stored as objects.
Data Complexity	RDBMS handles simple data.	OODBMS handles large and complex data.
Term	An entity refers to collection of similar items having same definition.	An class refers to group of objects having common relationships, behaviors and properties.
Data Handling	RDBMS handles only data.	OODBMS handles both data and functions operating on that data.
Objective	To keep data independent from application program.	To implement data encapsulation.
Key	A primary key identifies in object in a table uniquely.	Object Id, OID represents an object uniquely in group of objects.

Advantages of OODB

1. Can handle large collections of complex data including user defined data types.
2. Expressive data relationships
3. Version control for evolving classes and projects
4. Efficiently handles many-to-many relationships

Persistent Programming Language

Persistent Programming Language

- Programming languages that allow objects to continue existing after the program has been closed down are called persistent programming languages.
- A persistent programming language is extended with constructs to handle persistent data.
- In a persistent programming language:
  - ✓ *The query language is fully integrated with the host language and both share the same type system.*
  - ✓ *Any format changes required between the host language and the database are carried out transparently.*

Persistent Programming Language

- PPL allow an object to be created and stored in a database.
- PPL allows data to be manipulated directly from the programming language.
- No need to go through SQL.
- No need for explicit format changes.
- PPL allow objects to be manipulated in memory.
- No need to explicitly load from or store to the database.

Persistent Programming Language

Drawbacks:

1. It is easy to make programming errors that damage the database.
2. It is harder to do automatic high-level optimization.
3. They do not support declarative querying well.

Data Warehousing

Data Warehousing

- A Data Warehousing (DW) is the process for collecting and managing data from various sources to provide meaningful business insights.
- A Data warehouse is typically used to connect and analyze business data from heterogeneous sources.

How Datawarehouse Works?

- A Data Warehouse works as a central repository where information arrives from one or more data sources.
- Data may be:
  1. Structured
  2. Semi-structured
  3. Unstructured data
- The data is processed, transformed, and ingested so that users can access the processed data in the Data Warehouse through Business Intelligence tools, SQL clients, and spreadsheets.

Types of Datawarehouse

Three main types of Data Warehouses are:

1) Enterprise Data Warehouse:

2) Operational Data Store

3) Data Mart

Types of Datawarehouse

1) Enterprise Data Warehouse:

- Enterprise Data Warehouse is a Centralized Warehouse, which provides Decision Support Service across the enterprise.
- It offers a Unified Approach to Organizing and Representing Data.
- It also provides the ability to Classify Data according to the subject and Give Access according to those divisions.

Types of Datawarehouse

2) Operational Data Store:

- Operational Data Store (ODS) required when neither Data warehouse nor OLTP systems support organizations reporting needs.
- In ODS, Data warehouse is refreshed in real time.
- Hence, it is widely preferred for routine activities like storing records of the Employees.

Types of Datawarehouse

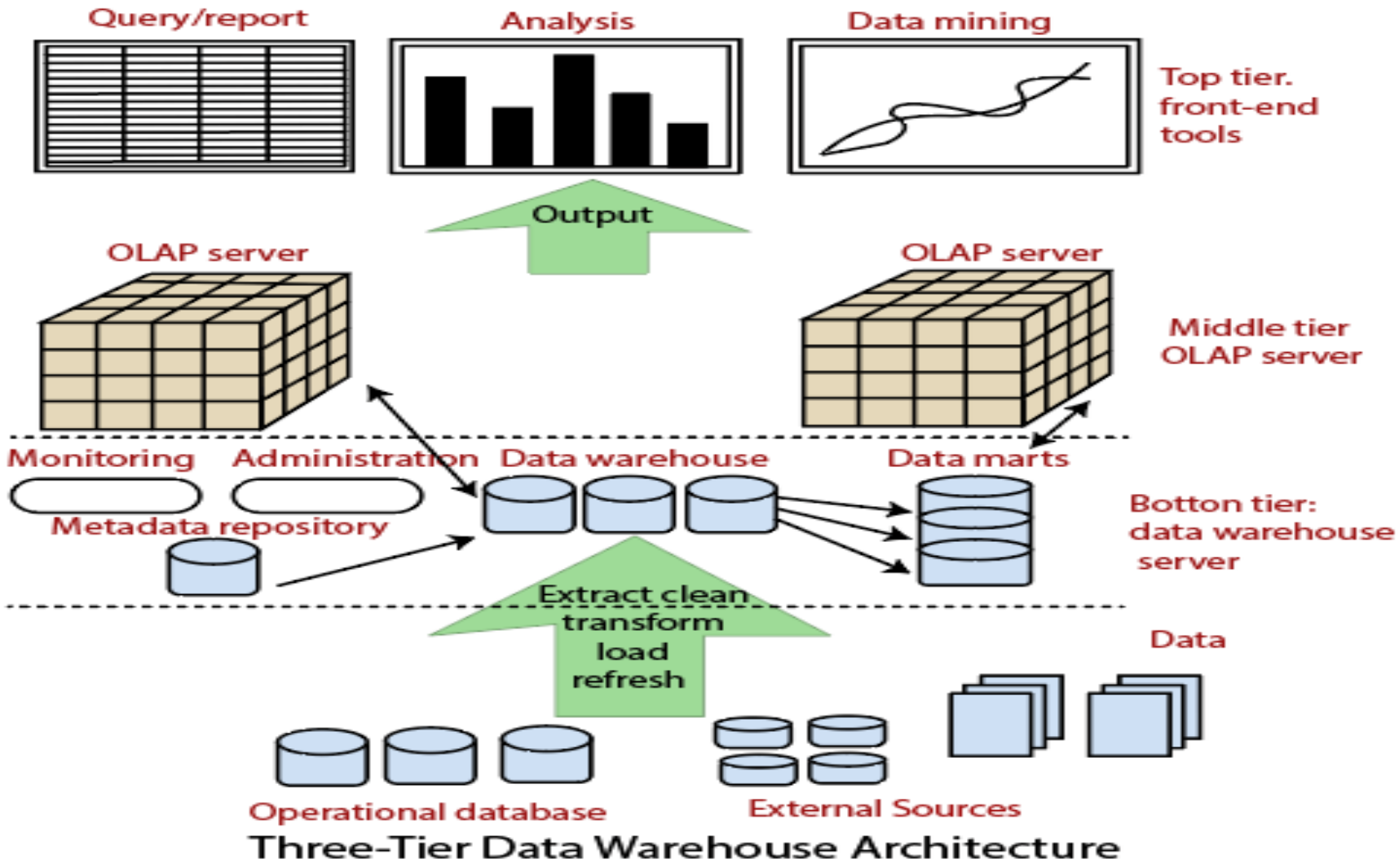
(3) Data Mart:

- A Data Mart is a subset of the data warehouse.
- It Specially Designed for a particular line of business, such as Sales, Finance, Sales or Finance.
- In an Independent Data Mart, data can collect directly from sources.

Datawarehouse Architecture

- Data Warehouse architecture is based on a Relational Database Management System Server that functions as the central repository for informational data.
- In the data warehouse architecture, Operational Data and Processing are separate from data warehouse processing.
- A Data Warehouse adopts a Three-tier Architecture.
  1. Bottom tier
  2. Middle tier
  3. Top tier

Datawarehouse Architecture



Three-Tier Data Warehouse Architecture

Datawarehouse Architecture

- **Bottom Tier: R**epresents the Data Warehouse Database Server. Back-end tools and utilities perform the Extract, Clean, Load, and Refresh functions.
- **Middle Tier: L**ies the OLAP Server. The ROLAP maps the operations on multidimensional data to standard relational OLAP (MOLAP) model, which directly implements the multidimensional data and operations.
- **Top-Tier:** Represents the Front-end Client Layer. This layer holds the Query Tools and Reporting Tools, Analysis Tools and Data Mining Tools.

Datawarehouse Architecture

Advantages:

1. Data warehouse allows business users to quickly access critical data all in one place.
2. Data warehouse provides consistent information.
3. It is also supporting ad-hoc reporting and query.
4. Data Warehouse helps to integrate many sources of data to reduce stress on the production system.
5. Data warehouse helps to reduce total turnaround time for analysis and reporting.
6. Restructuring and Integration make it easier for the user to use for reporting and analysis.

Datawarehouse Architecture

Disadvantages:

1. Creation and Implementation of Data Warehouse is surely time confusing affair.
2. Difficult to make changes in data types, ranges, data source schema, indexes, and queries.
3. It is too complex for the average users.
4. Organizations need to spend lots of their resources for training and Implementation purpose.

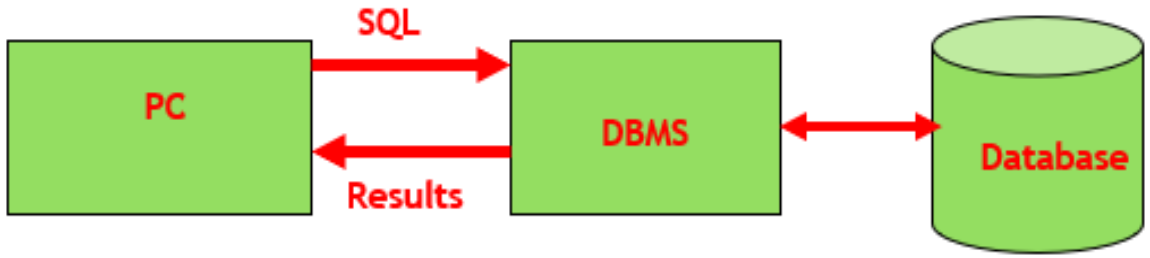
DATA MINING

DATA MINING

- There is a huge amount of data available in the Information Industry. This data is of no use until it is converted into useful information.
- To analyze this huge amount of data and extract useful information from it.
- Extraction of information is not only the single process, it involves Data Cleaning, Data Integration, Data Transformation, Data Mining, Pattern Evaluation and Data Presentation.
- Once all these processes are over, we would be able to use this information in many applications such as Fraud detection, Market analysis, Science exploration, etc.

WHAT IS DATA MINING?

- "Data Mining" refers to the extraction of useful information from a bulk of data or data warehouses.



WHAT IS DATA MINING?

- Data Mining, the result of extraction process is not data!! Instead, the result of data mining is the patterns and knowledge that we gain at the end of the extraction process.
- In that sense, Data Mining is also known as Knowledge Discovery in Databases (KDD) or Knowledge Extraction.

WHO USES DATA MINING?

- Data Mining is used in almost all the places where a large amount of data is stored and processed.
- Example, Banks typically use ‘data mining’ to find out their prospective customers who could be interested in Credit Cards, Personal Loans or Insurances as well.
- Since banks have the Transaction Details and Detailed Profiles of their customers, they analyze all this data and try to find out patterns which help them predict that certain customers could be interested in personal loans etc.

PURPOSE OF DATA MINING?

- The Data Mining helps to Predict Hidden Patterns, Future Trends and Behaviors and allowing businesses to Take Decisions.
- Technically, data mining is the computational process of Analyzing Data from different perspective.
- Data Mining can be applied to any type of data e.g. Data Warehouses, Transactional Databases, Relational Databases, Multimedia Databases, Spatial Databases, Time-series Databases, World Wide Web.

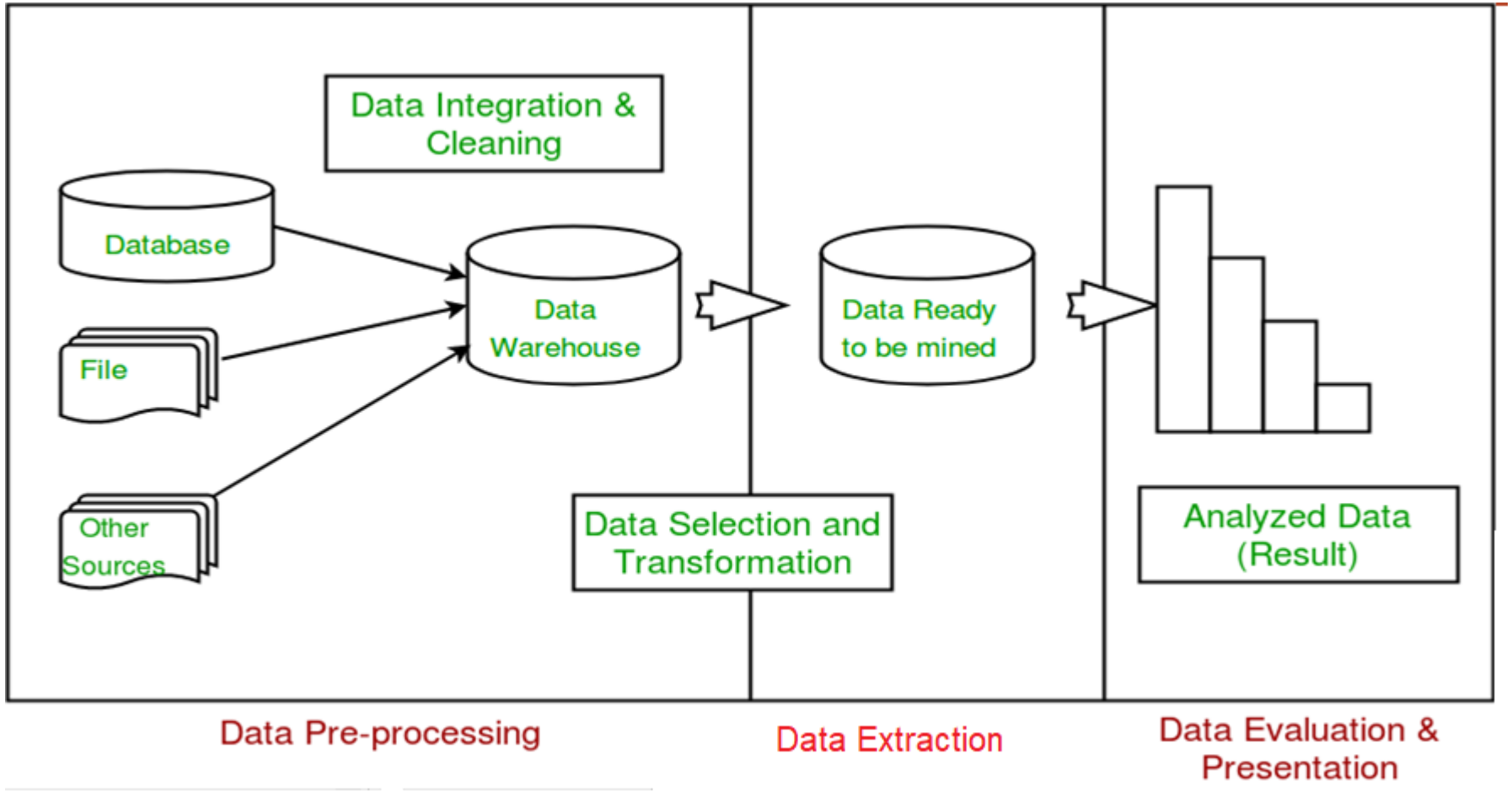
Data Mining as a Whole Process

The whole process of Data Mining comprises of Three Main Phases:

1. Data Pre-processing – *Data Cleaning, Integration, Selection and Transformation.*
2. Data Extraction – *Occurrence of exact data mining*
3. Data Evaluation and Presentation – *Analyzing and presenting results*

Data Mining as a Whole Process

The whole process of Data Mining comprises of Three Main Phases:



(1) Data Pre-Processing

- Data Pre-Processing is a data mining technique which is used to transform the raw data in a useful and efficient format.
- Steps involved in Data Pre-Processing,
  1. Data Cleaning.
  2. Data Transformation
  3. Data Reduction.

(1) Data Pre-Processing

1. **Data Cleaning:**

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done.

2. **Data Transformation:**

To transform the data in appropriate forms suitable for mining process.

3. **Data Reduction:**

While working with huge volume of data, analysis became harder.

In order to get rid of this, we uses data reduction technique.

It aims to increase the storage efficiency and reduce data storage and analysis costs.

(2) Data Extraction

- In this Phase, Mathematical Models are used to determine Data Patterns.
- Based on the Business Objectives, suitable Modeling Techniques should be selected for the Prepared Dataset.
- Create a scenario to test check the Quality and Validity of the model.
- Run the model on the prepared dataset.
- To make sure that model can Meet Data Mining Objectives.

(3) Data Evaluation & Presentation

Evaluation:

- In this phase, Patterns identified are Evaluated against the business objectives.
- Gaining Business Understanding is an iterative process.
- A Go or No-Go Decision is taken to move the model in the deployment phase.

(3) Data Evaluation & Presentation

Presentation:

- Ship your data mining discoveries to everyday business operations.
- The knowledge or information discovered during data mining process should be made easy to understand for non-technical stakeholders.
- A detailed deployment plan, for shipping, maintenance, and monitoring of data mining discoveries is created.
- This helps to improve the organization's business policy.

Applications of Data Mining

1) Communications.
2) Insurance.
3) Education.
4) Manufacturing.
5) Banking.
6) Retail.
7) Service Providers.
8) E-Commerce.
9) Super Markets.
10) Crime Investigation.
11) Bioinformatics'

Benefits of Data Mining

1) Helps companies to get Knowledge-Based Information.
2) Helps organizations to make the profitable adjustments in operation and production.
3) The data mining is a cost-effective and efficient solution.
4) Data mining helps with the decision-making process.
5) Facilitates automated prediction of trends and behaviors.

Disadvantages of Data Mining

1) There are chances of companies may sell useful information of their customers to other companies for money. For example, American Express has sold credit card purchases of their customers to the other companies.
2) Many data mining analytics software is difficult to operate and requires advance training to work on.
3) The selection of correct data mining tool is a very difficult task.