23CSE241T - DATABASE MANAGEMENT SYSYTEM

LECTURE NOTES

B.TECH II YEAR – IV SEM (R23)

DEPARTMENT OF CSM

SRINIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES

(Autonomous Approved by AICTE, New Delhi. Affiliated to JNTUA, Ananthapuramu)

Prepared: V.Ragubathy

SYLLABUS

UNIT-1: INTRODUCTION TO DATABASE MANAGEMENT SYSTEM AND ENTITY RELATIONSHIP MODEL

Database system - Characteristics (Database Vs File System) - Database Users - Advantages of Database systems - Database applications - Brief introduction of different Data Models - Conceptsof Schema - Instance and data independence -Three tier schema architecture for data independence - Database system structure environment - Centralized and Client Server architecture for the database - Introduction to Entity Relationship Model - Representation of entities - Attributes - Entity set – Relationship - Relationship set – Constraints - Sub classes - super class - Inheritance- Specialization -Generalization using ER Diagrams.

UNIT –2: RELATIONAL MODEL

Introduction to Relational model - Concepts of domain – Attribute – Tuple - Relation importance ofnull values - Constraints (Domain, Key constraints, integrity constraints) and their importance - Relational Algebra, Relational Calculus - BASIC SQL: Simple Database schema - Data Base Language - types- Table definitions (create, alter), different DML operations (insert, delete, update).

UNIT –3: INTRODUCTION TO STRUCTURED QUERY LANGUAGE

Basic SQL querying (select and project) using where clause arithmetic & logical operations - SQL functions(Date and Time, Numeric, String conversion) - Creating tables with relationship, Implementation of key and integrity constraints - Nested queries, sub queries, grouping, aggregation, ordering - Implementation of different types of Joins, view (updatable and non- updatable) - Relational set operations.

UNIT –4: NORMALIZATION

Purpose of Normalization and schema refinement - Concept of functional dependency - normal forms based on functional dependency - Lossless join and dependency preserving decomposition (1NF, 2NF and 3 NF), concept of surrogate key - Boyce-Codd normal form(BCNF) - MVD - Fourth normal form(4NF) - Fifth Normal Form (5NF).

UNIT –5: TRANSACTION CONCEPT AND INDEXING CONCEPTS

Transaction State - ACID properties - Concurrent Executions – Serializability - Recoverability, Implementation of Isolation - Testing for Serializability - Lock based - Time stamp based optimistic - Concurrency protocols – Deadlocks - Failure Classification - Storage, Recovery and Atomicity - Recovery algorithm - Introduction to Indexing Techniques - B+ Trees, operations onB+Trees - Hash Based Indexing

Prepared: V.Ragubathy

UNIT-I

Introduction:

In computerized information system data are the basic resource of the organization. So, proper organization and management for data is required for organization to run smoothly. Database management system deals the knowledge of how data stored and managed on a computerized information system. In any organization, it requires accurate and reliable data for better decision making, ensuring privacy of data and controlling data efficiently. The examples include deposit and/or withdrawal from a bank, hotel, airline or railway reservation, purchase items from supermarkets in all cases, a database is accessed

What is data?

Data are the known facts or figures that have implicit meaning. It can also be defined as it is the representation of facts, concepts or instructions in a formal manner, which is suitable for understanding and processing. Data can be represented in alphabets (A-Z, a-z), digits (0-9) and using special characters (+,-.#,$, etc) e.g: 25, "ajit" etc.

DATA: - Any factor that can be stored.

- Data is collection of raw facts such
- as numbers, words, observations etc..

Example: text, numbers, images, videos and speech.

What Is a DBMS?

A Database Management System (DBMS) is a software package designed to interact with end- users, other applications, store and manage databases. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.

DBMS contains information about a particular enterprise

- Collection of interrelated data
- Set of programs to access the
- An environment that is both *convenient* and *efficient* to use

Why Use a DBMS?

A database management system stores, organizes and manages a large amount of information within a single software application. It manages data efficiently and allows users to perform multiple tasks with ease.

- Reduced application development time.

- Data integrity and security.
- Uniform data administration.
- Concurrent access, recovery from crashes

PURPOSE OF DATABASE SYSTEM

In the early days, database applications were built directly on top of file systems. A DBMS provides users with a systematic way to create, retrieve, update and manage data. It is a middleware between the databases which store all the data and the users or applications which need to interact with that stored database. A DBMS can limit what data the end user sees, as well as how that end user can view the data, providing many views of a single database schema.

Database + database management system = database system

File: The collection of data is called a file that is stored in the computer with a file name. These files are managed by the File Based system.

File Based Systems

A file system is a technique of arranging the files in a storage medium like a hard disk, pen drive, DVD, etc. It helps you to organizes the data and allows easy retrieval of files when they are required.

Drawbacks of using file systems to store data:

1. Data Redundancy and Inconsistency

- Same data may be stored in multiple files → leads to duplication.
- If one file is updated and others are not → inconsistency occurs.

Example: A student's address stored in 5 files; one file is updated, others are not.

2. No Data Sharing / Multi-user Access Problems

- File systems do not support multiple users accessing or updating the same file safely.

3. Poor Data Security

- File systems cannot provide fine-grained security.

Example: You cannot easily restrict access only to a particular field like salary.

4. Lack of Data Integrity

- File systems cannot enforce rules automatically.

Example: No check for unique roll numbers or valid age.

5. No Backup and Recovery

- If files are accidentally deleted or system crashes → data may be lost
- File systems provide **very limited or no recovery** support.

6. Difficult to Access Data

- You must write long programs to search or retrieve data.
- No query language like SQL.

7. No Concurrency Control

- Two users modifying the same file may cause **loss of data** or **corrupted files**.

8. Poor Scalability

- Managing very large data sets through files is complex and slow.

Difference between File System and DBMS:

Basis	File System	DBMS
Structure	The file system is software that manages and organizes the files in a storage medium within a computer.	DBMS is software for managing the database.
Data Redundancy	Redundant data can be present in a file system.	In DBMS there is no redundant data.
Backup and Recovery	It doesn't provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.
Query processing	There is no efficient query processing in the file system.	Efficient query processing is there in DBMS.
Consistency	There is less data consistency in the file system.	There is more data consistency because of the process of normalization.
Complexity	It is less complex as compared to DBMS.	It has more complexity in handling as compared to the file system.
Security Constraints	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file systems.
Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
Data Independence	There is no data independence.	In DBMS data independence exists.
User Access	Only one user can access data at a time.	Multiple users can access data at a time.

Database Users

- People who work with a database can be:
  (a) Database users
- Database users are the persons who interact with the database and take the benefits of the DB.
- Users are differentiated by the way they interact with the system.

Four types of DB users are:

1. **Naive users / Native users / End users**
2. **Application programmers**
3. **Sophisticated users**
4. **Specialized users**

1) Naive Users / Native Users / End Users

- Unsophisticated users who use the existing applications to interact with the database.

  Example: People who use online applications like reserving flight tickets, movie tickets, etc.

2) Application Programmers

- Computer professionals who write the application programs.
- They interact with the database through **DML queries**.

Example:

Online application or standalone application developers who write queries in their applications to interact with the DB.

3) Sophisticated Users

- People who interact directly with the database by writing SQL queries.
- These people are called sophisticated users.

Example:
Analysts who submit SQL queries to explore data in the DBMS.

4) Specialized Users

- They are also sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
- Developers who develop complex database applications.

Example:
Computer-Aided Design (CAD) systems that need to store complex data types like graphics data, audio data, etc.

Advantages of Database Management System
The advantages of database management systems are:

1. **Data Security:**
   - DBMS enhances data security through access control and encryption.
   - It enforces privacy policies and prevents unauthorized access.

2. **Data integration:**
    - DBMS unifies data from different sources into a centralized system. This provides a coherent organizational view of operations.
    - It helps to keep track of how one segment of the company affects another segment.
3. **Data abstraction:**
    - Since many complex algorithms are used by the developers to increase the efficiency of databases that are being hidden by the users through various data abstraction levels to allow users to easily interact with the system.
4. **Reduction in data Redundancy:**
    - DBMS avoids data duplication by enforcing unique constraints.
    - It removes unnecessary repetitive entries in databases. This ensures efficient use of storage and improves consistency.

      Example - If there are two same students in different rows, then one of the duplicate data will be deleted.
5. **Data sharing:**
    - A DBMS provides a platform for sharing data across multiple applications and users, which can increase productivity and collaboration.
6. **Data consistency and accuracy:**
    - DBMS enforces integrity constraints to maintain valid data.
    - It minimizes discrepancies by syncing updates across all views. This reduces errors and ensures reliable & consistent data.
7. **Data organization:**
    - A DBMS provides a systematic approach to organizing data in a structured way, which makes it easier to retrieve and manage data efficiently.
8. **Efficient data access and retrieval:**
    - DBMS allows for efficient data access and retrieval by providing indexing and query optimization techniques which reduces time taken to retrieve large datasets.
    - It boosts system performance and user satisfaction.
9. **Concurrency and maintained Atomicity**:
    - That means, if some operation is performed on one particular table of the database, then the change must be reflected for the entire database.
    - The DBMS allows concurrent access to multiple users by using the synchronization technique.

10. **Scalability and flexibility:**
    - DBMS is highly scalable and can easily accommodate changes in data volumes and user requirements.
    - It allows flexible schema modifications and expansion.
    - This makes it ideal for dynamic organizational environments and can scale up or down depending on the needs of the organization.

Database Applications

Database applications are software systems that use a **database** to store, retrieve, update, and manage data efficiently. They are used wherever large volumes of data must be handled accurately, securely, and quickly.

1. Banking Systems

Databases manage all financial transactions and customer details.

- Customer account information
- Deposits and withdrawals
- ATM and online banking transactions
- Loan and credit card management

2. Airline / Railway Reservation Systems

Databases support real-time ticket booking and seat allocation.

- Passenger details
- Seat availability
- Ticket booking and cancellation
- Schedule and fare management

3. Education and University Management

Databases store academic and administrative records.

- Student details
- Course registration
- Attendance and exam results
- Fee management

4. Healthcare and Hospital Management

Databases maintain patient and medical data.

- Patient records
- Doctor schedules
- Medical history and lab reports
- Billing information

5. E-Commerce Applications

Online shopping platforms depend on databases.

- Product catalog
- Customer accounts
- Orders and payments
- Inventory and delivery tracking

6. Telecommunication Systems

Databases manage subscribers and billing.

- Customer information
- Call records
- Usage data
- Billing and recharge details

7. Government and Public Administration

Databases help manage citizen and service data.

- Census information
- Tax records
- Passport and ID systems
- Voter registration

8. Social Media Applications

Databases handle large amounts of user-generated data.

- User profiles
- Posts, comments, and likes
- Messages and connections

9. Manufacturing and Inventory Systems

Databases track production and stock.

- Raw materials
- Production planning
- Sales and supplier details

10. Library Management Systems

Databases organize library resources.

- Book catalog
- Member records
- Issue and return details
- Fine calculation

Brief Introduction of Different Data Models

A **data model** is a collection of concepts used to describe the **structure of a database**, the **relationships** among data, and the **constraints** on data. Data models help in designing databases at different levels of abstraction.

1. Hierarchical Data Model

- Organizes data in a **tree-like structure**.
- Each parent can have **multiple children n**, but a child has **only one parent**.
- Relationships are **one-to-many**.
- Data is accessed by navigating from the root.
- **Example:** Organization chart, file systems.
- **Limitation:** Difficult to represent many-to-many relationships.

2. Network Data Model

- Data is represented as a **graph structure**.
- A record can have **multiple parents and children**.
- Supports **many-to-many relationships**.
- Uses pointers to establish relationships.
- **Example:** Complex database applications.
- **Limitation:** Complex structure and difficult to maintain.

3. Relational Data Model

- Data is organized in **tables (relations)** consisting of rows and columns.
- Each table has a **primary key**.
- Relationships are created using **foreign keys**.
- Uses **SQL** for data manipulation.
- **Example:** MySQL, Oracle, PostgreSQL.
- **Advantages:** Simple, flexible, widely used.

4. Entity–Relationship (ER) Model

- Used for **database design**, not implementation.
- Represents data as **entities**, **attributes**, and **relationships**.
- Visualized using **ER diagrams**.

Prepared: V.Ragubathy

- Helps in understanding database requirements.
- **Example:** Student–Course relationship.

5. Object-Oriented Data Model

- Data is stored as **objects**, similar to object-oriented programming.
- Objects contain **data and methods**.
- Supports **inheritance, encapsulation, and polymorphism**.
- **Example:** Object-oriented databases.
- **Advantage:** Suitable for complex data like multimedia.

6. Object-Relational Data Model

- Combination of **relational** and **object-oriented** models.
- Supports complex data types in relational tables.
- Used in advanced DBMS.
- **Example:** PostgreSQL, Oracle.

7. Semi-Structured Data Model

- Data does not follow a strict table structure.
- Schema is **flexible or partially defined**.
- Represented using **XML, JSON**.
- **Example:** Web data, NoSQL databases.

8. Physical Data Model

- Describes **how data is stored** in the database.
- Includes indexes, storage structures, and access paths.
- Used by database administrators.

Instance and Data Independence

In a **Database Management System (DBMS)**, ***instance*** and ***data independence*** are core concepts that explain how data is stored, viewed, and modified without affecting users or applications.

1. Instance

An **instance** is the **actual data stored in the database at a particular moment of time**.

- It represents the **current state** of the database.
- It changes frequently as data is **inserted, updated, or deleted**.

Example

Consider a **Student** table:

Roll No	Name	Marks
101	Ravi	85
102	Anu	90

This table content at this moment is an **instance**.

If a new student is added or marks are updated, the **instance changes**, but the schema remains the same.

2. Data Independence

Data independence is the **ability to change the schema at one level of the database system without affecting the schema at the next higher level**.

It helps in:

- Reducing application maintenance
- Improving flexibility
- Ensuring long-term usability of applications

DBMS supports data independence using the **three-level architecture**:

- Physical level
- Logical level
- View level

Concepts of Schema

The **schema** is the **logical structure of the database**, also called the **database schema**. It defines how the data in the database is organized and related.

A schema is **analogous to the type information of a variable in a program**, which defines the structure but not the actual values.

There are **three types of schema**:

1. Physical schema
2. Logical schema
3. View schema

Example

A database may consist of information about a **set of customers**, **accounts**, and the **relationship between them**.

The structure that defines these entities and their relationships is called the **schema**.

Physical Schema

- Database design at the **physical level** is called the **physical schema**.
- It describes **how data is stored physically** in storage devices.
- This level explains how data is stored in **blocks of storage**, files, and indexes.

Logical Schema

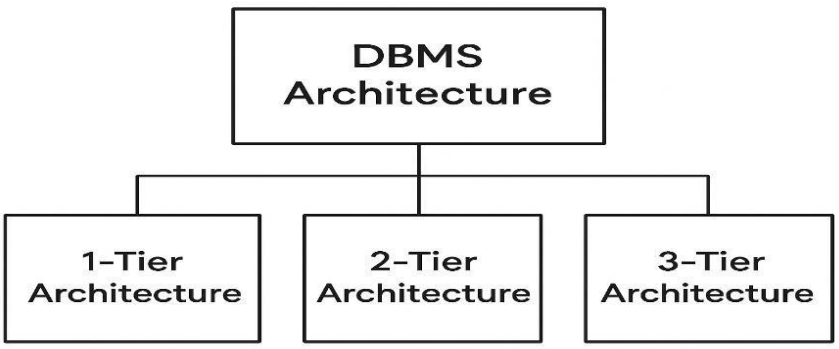
- Database design at the **logical level** is called the **logical schema**.
- Programmers and database administrators work at this level.
- At this level, data is described in terms of **data records and data structures**.
- The **internal details of data structure implementation are hidden** at this level.

View Schema

- Database design at the **view level** is called the **view schema**.
- This level describes how **end users interact with the database system**.
- Different users may have different views of the same database.

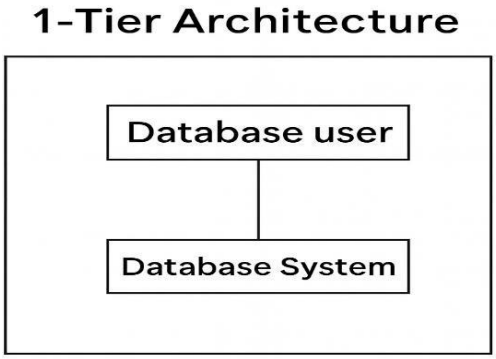
Database Architecture

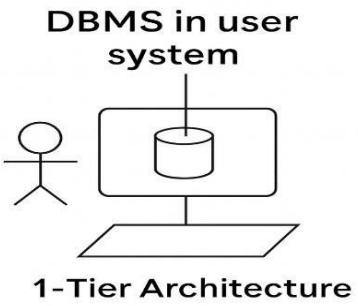
- DBMS Architecture depends upon
    1. the underlying computer system on which database system runs
    2. how users are connected to the database to get their request done.
- DBMS architecture can be seen as a single tier, 2-tier or 3-tier architecture.



1- Tier Architecture

- Here, DBMS is stored directly in the user system.
- User can interact directly through interfaces like SQL.
- No network connection is required to perform actions on the database.



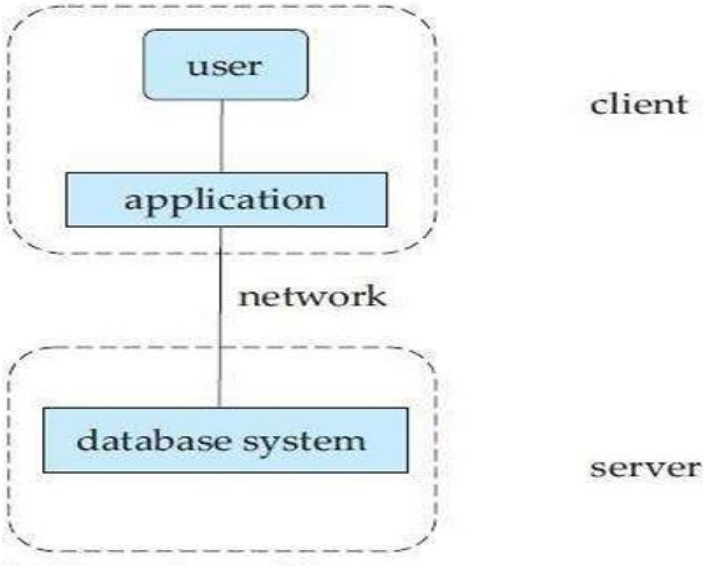


1-Tier Architecture is used

- Where data does not change frequently and where no multiple user is accessing the system.
- Such architecture is rarely used in production.

2- Tier Architecture

- In 2-tier architecture, applications on the client end can directly communicate with the database at the server side.
- An Application Programming Interface (APIs) like **ODBC** or **JDBC** are used by client-side programs to call the DBMS.
- The 2-tier architecture is used inside an organization, where clients access the database server directly.
- **Example:** Railway reservation system, where the clerk acts as a client and accesses the railway server directly.



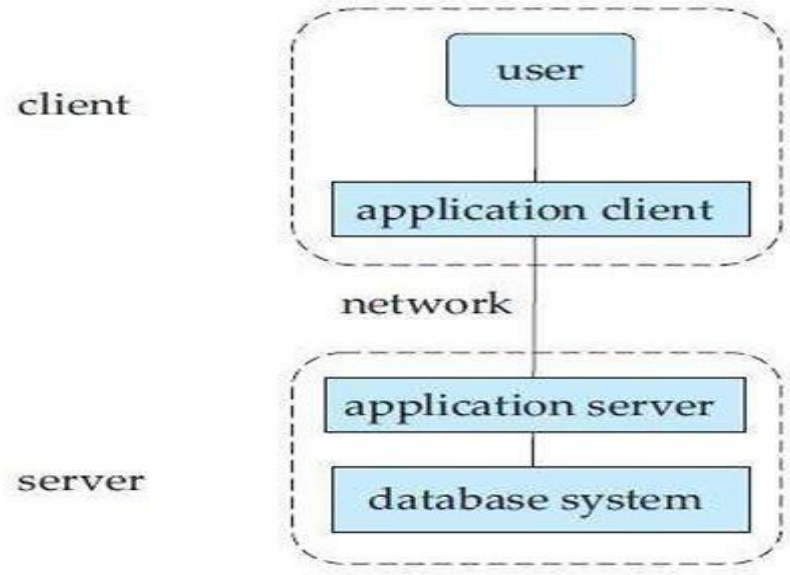
+

Disadvantages

- Scalability, i.e., it gives poor performance when there are a large number of users.
- Less secure as client can access the server directly.

3- Tier Architecture

- The 3-tier architecture contains another layer of application server between the client and server.
- In this architecture, client can't directly communicate with the server (DB server).
- The application on the client side interacts with an application server which further communicates with the database system and then the query processing and transaction management take place.
- The intermediate layer of application server acts as a medium for exchange of partially processed data between server and client.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.



- 3-tier architecture is used in large web application.
- It is the most popular DBMS architecture.

Summary

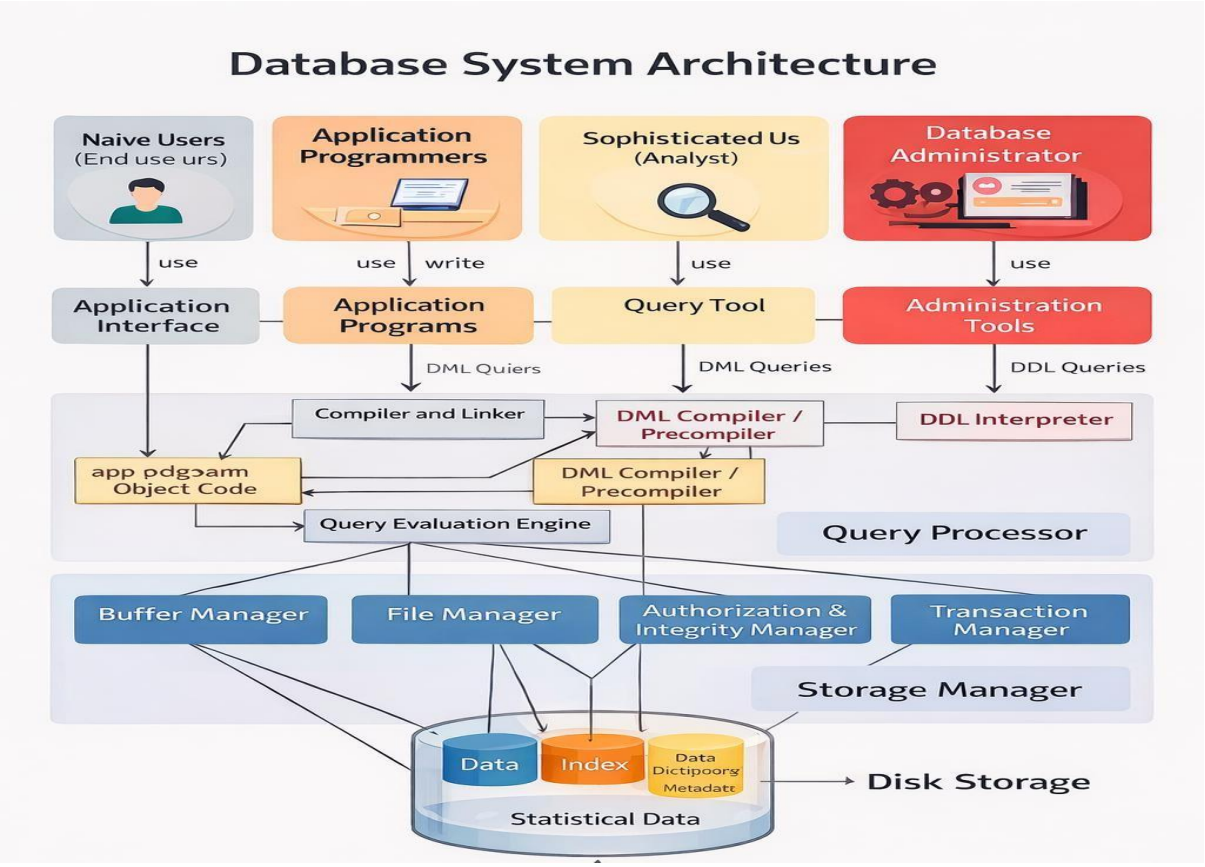
- If a **single user** wants to use a DBMS, they should go for **1-tier architecture**.
- If **multiple users** want to use a DBMS, then deploy it on a **server** so that multiple clients can access the database, i.e., **2-tier architecture**.

  Example: Students accessing DBMS from a server.
- In **2-tier architecture**, there is **no proper security**. Anybody can access the stored data.
- If **multiple users** want to use DBMS in a **secured way**, go for **3-tier architecture**.
- In **3-tier architecture**, the user interacts with the database **through an application**. Data can be accessed **according to application design** and cannot be accessed beyond the application scope.

Database System Structure

- DBMS acts as an **interface between the user and the database**.
- The user requests the DBMS to perform various operations such as:
  - Insert
  - Delete
  - Update
  - Retrieval
- DBMS structure is **partitioned into modules** for different functions.
- DBMS is divided into **two functional components**:

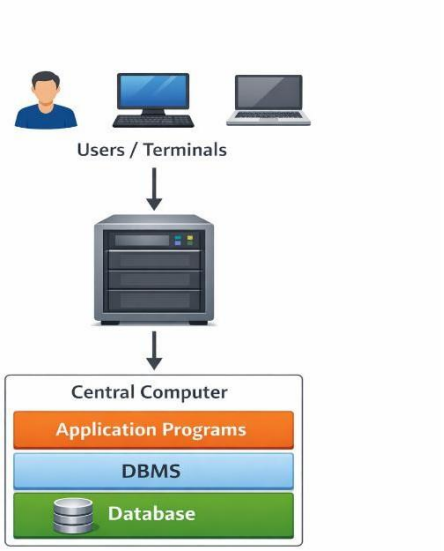
1) **Query Processor**
2) **Storage Manager**



Centralized Database Architecture

In **centralized architecture**, the **entire database system** (database, DBMS, and application programs) is located at **one central computer**.

All users access the database from terminals connected to this central system.



- All data is stored at **one location**.
- Users send requests to the central system.
- The DBMS processes the request and sends results back to users.

Advantages

- Easy to **manage and control**
- Strong **data security**
- Easy **backup and recovery**
- Data **consistency** is maintained

Disadvantages

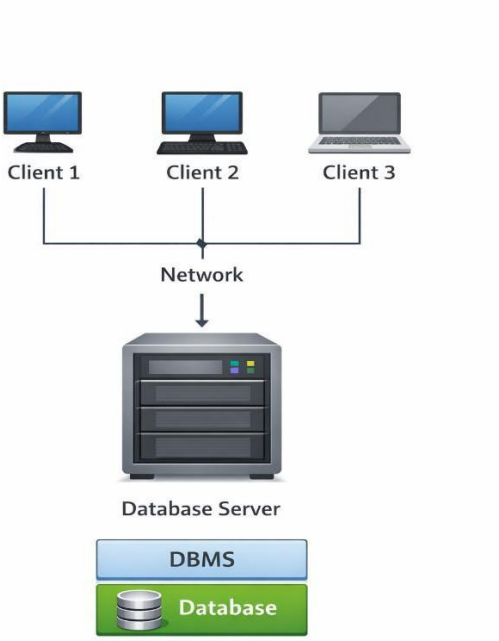
- Single point of failure
- Performance degrades as users increase
- Not suitable for large or geographically distributed users

Example

- Mainframe-based banking systems
- University database running on a single server

Client–Server Database Architecture

In **client–server architecture**, the **database and DBMS** are stored on a **server**, while users interact with the system using **client machines**.



Working

- Clients send requests (SQL queries) to the server.
- The server processes the request.
- Results are sent back to the client.

Types of Client – Server Architecture

a) Two-Tier Architecture

- Client: User interface + application logic
- Server: DBMS + database

Ex: Client <----> Database Server

b) Three-Tier Architecture

- Presentation layer (Client)
- Application server
- Database server

Ex: Client → Application Server → Database Server

Advantages

- Better **performance** than centralized systems
- **Scalable** (supports more users)
- Efficient use of network
- Suitable for distributed environments

Disadvantages

- Server failure affects all clients
- Network dependency
- More complex than centralized systems

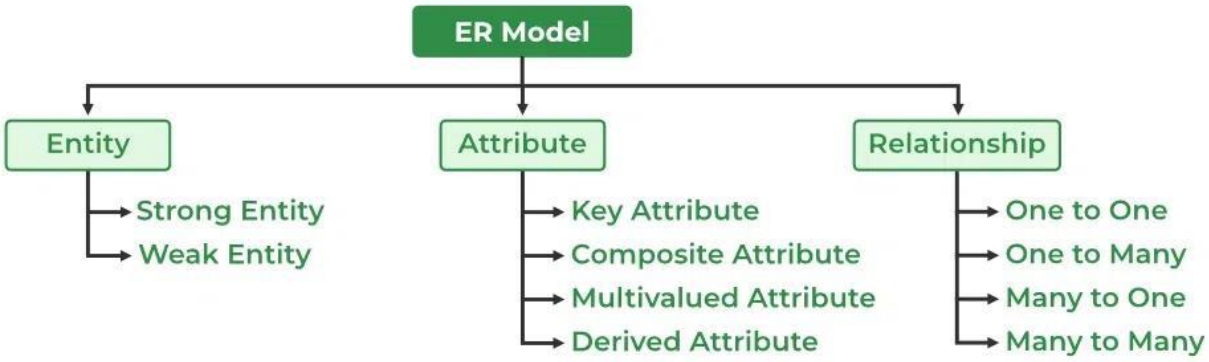
Example

- Online banking systems
- Web applications (Amazon, Flipkart, etc.)
- Enterprise applications

Introduction of ER Model

The Entity-Relationship Model (ER Model) is a conceptual model for designing a databases. This model represents the logical structure of a database, including entities, their attributes and relationships between them.

 Entity: An objects that is stored as data such as *Student*, *Course* or *Company*.
 Attribute: Properties that describes an entity such as *StudentID*, *CourseName*, or *EmployeeEmail*.
 Relationship: A connection between entities such as "a *Student* enrolls in a *Course*".

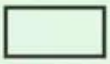
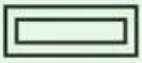


Why Use ER Diagrams In DBMS?

 ER diagrams represent the E-R model in a database, making them easy to convert into relations (tables).
 These diagrams serve the purpose of real-world modeling of objects which makes them intently useful.
 Unlike technical schemas, ER diagrams require no technical knowledge of the underlying DBMS used.

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

The main components of the ER Model are:

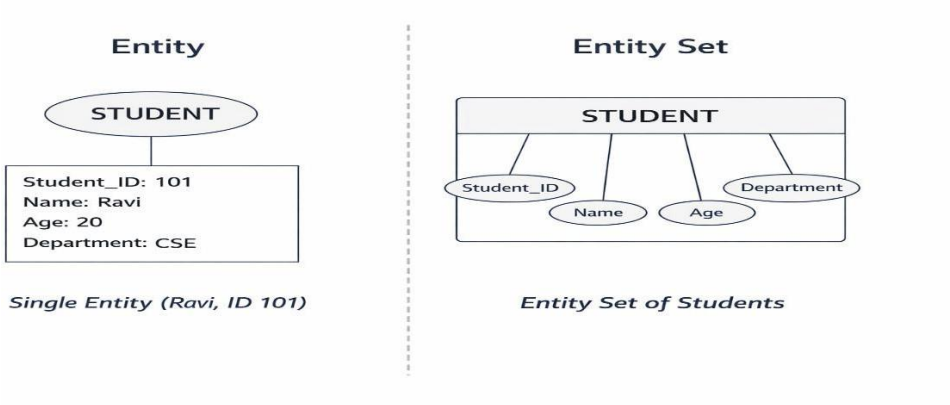
1. Entity sets
2. Relationship sets
3. Attributes

Entity

An **Entity** is a real-world object that is distinguishable from other objects. In a database, an entity is described using a set of attributes.

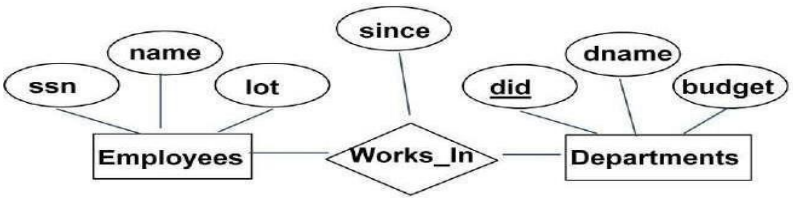
1) Entity Set

- An **Entity Set** is a collection of similar entities.
- For example, all employees in an organization form an entity set called *Employee*.
- All entities in an entity set have the same set of attributes, until ISA (Inheritance) hierarchies are considered.
- Each entity set has a **key**, which uniquely identifies each entity in the set.
- Every attribute has a **domain**, which specifies the set of possible values that the attribute can take.
- Entity sets are represented by **rectangles** in an ER diagram.



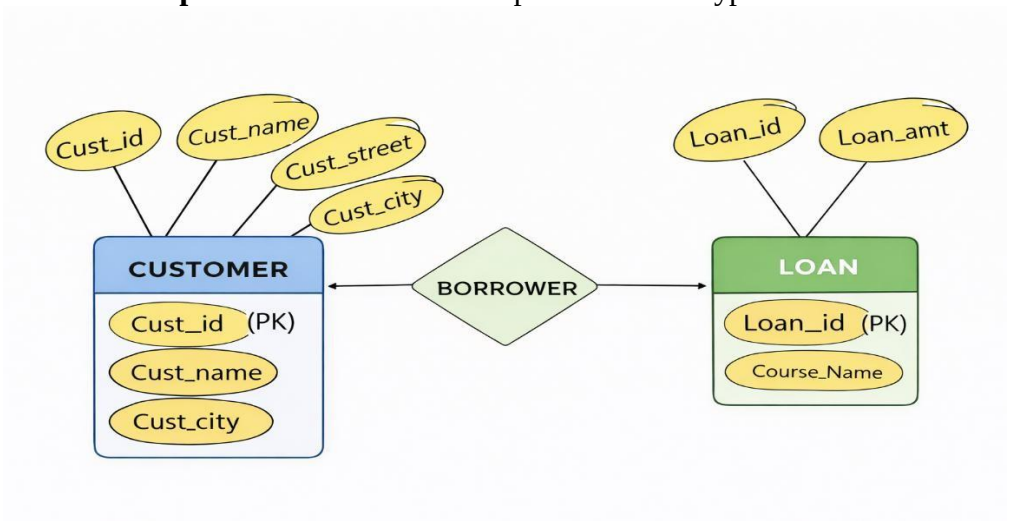
3)Relationship Set

Relationship: Association among two or more entities. E.g., Attishoo works in Pharmacy department.



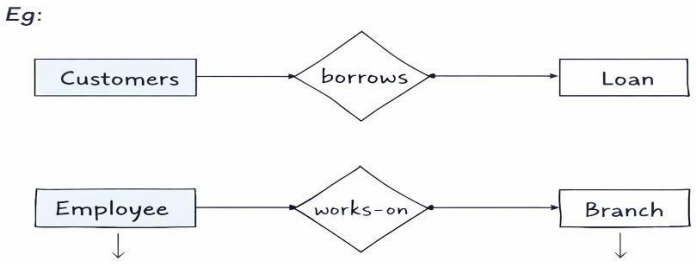
- A **relationship** is an association among several entities.

- A **relationship set** is a set of relationships of the same type.



- Diamonds are used to represent relationship sets in ER diagrams.

Eg:



3) Attribute

- Entities are represented by means of their properties, called attributes.
- All attributes have values. For example, a student entity may have name, ID, age attributes.
- An attribute describes the properties of an entity.

An attribute can be categorized as:

1. Simple and Composite attributes
2. Single-valued and Multivalued attributes
3. Derived attributes

Simple Attribute

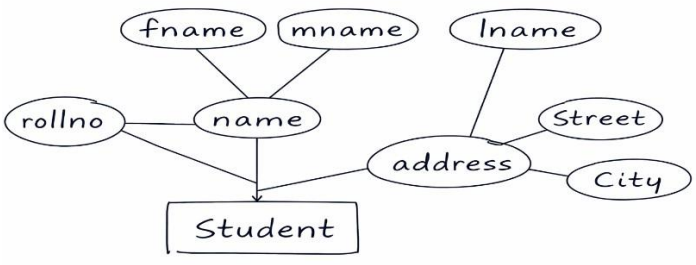
- Simple attributes are **atomic values**, which **cannot be divided further**.
- For example:
    - A student's **phone number** is an atomic value of 10 digits.
    - A **student roll number** cannot be further divided.

Composite Attribute

- An attribute which can be divided further.

  Eg: Student name can be further divided into **first name, middle name, and last name**.

- **Address** is another example of a composite attribute.



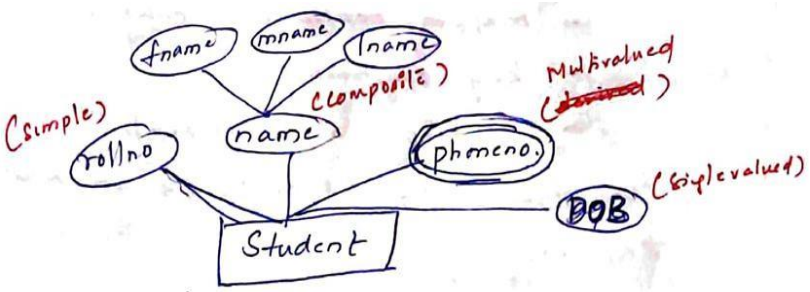
Single-Valued Attribute

- A **single-valued attribute** is an attribute that can hold **only one value**.
- Example:
  - The **age** attribute can hold only **one value at a time**.

Multivalued Attribute

- A **multivalued attribute** is an attribute that can hold **multiple values**.
- It is **represented using a double oval** in an **ER diagram**.

Example: A person can have **more than one phone number**, so **phone numbers** are a **multivalued attribute**.

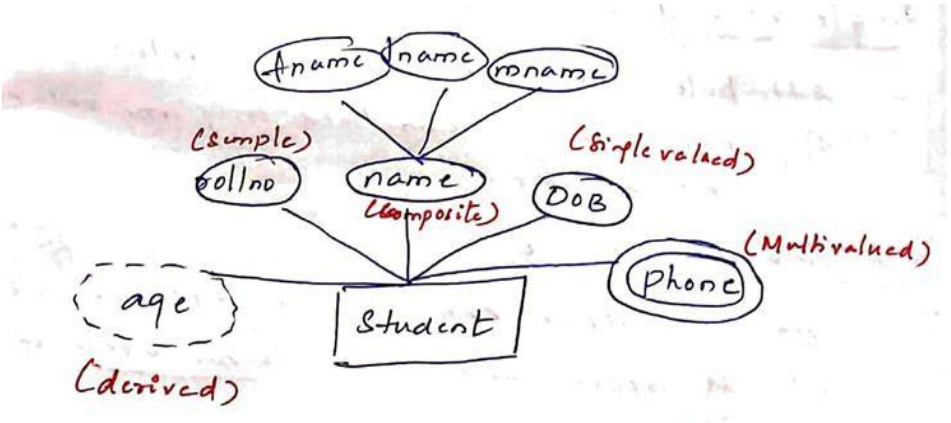


Derived Attribute

- A **derived attribute** is an attribute whose value is **derived from another related attribute**.
- Example:
  - The **age** attribute can be derived from **Date of Birth (DOB)**.

Age = Current Date − DOB

- The age attribute is **derived from DOB**, so it does not need to be stored separately.
- In an **ER diagram**, a derived attribute is **represented using a dashed oval**.



Prepared: V.Ragubathy

Constraints
An ER schema may define:

- Mapping cardinalities
- Key constraints
- Participation constraints

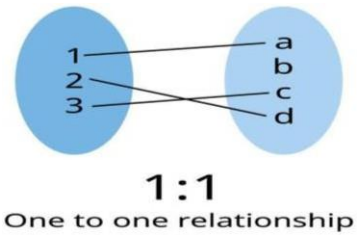
to which the content of the database must conform.

Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set.
- For a binary relationship set **R** between entity sets **A** and **B**, the mapping cardinalities must be one of the following:

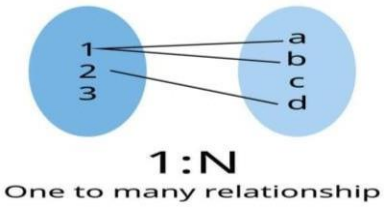
1) One-to-One (1 : 1)

- An entity in **A** is associated with **at most one** entity in **B**, and
- An entity in **B** is associated with **at most one** entity in **A**.



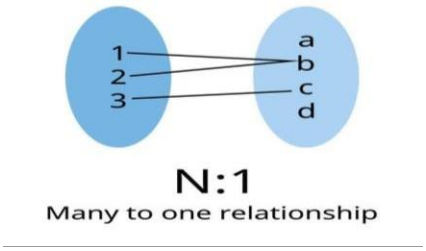
2) One-to-Many (1 : N)

- An entity in **A** is associated with **any number of entities** in **B**, and
- An entity in **B** is associated with **at most one** entity in **A**.



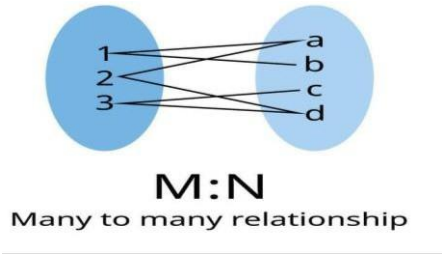
3) Many-to-One (N : 1)

- An entity in **A** is associated with **at most one** entity in **B**, and
- An entity in **B** is associated with **any number of entities** in **A**.



4) Many-to-Many (M : N)

- An entity in **A** is associated with **any number of entities** in **B**, and
- An entity in **B** is associated with **any number of entities** in **A**.



Keys Constraints

A **key** is an attribute or set of attributes that uniquely identifies any record from the table.

Purpose

- A key is used to uniquely identify the tuple.
- It is also used to establish and identify relationships between tables.

Types of Keys

1. Super Key
2. Candidate Key
3. Primary Key
4. Alternate Key
5. Foreign Key
6. Composite Key

Super Key

- It is a combination of all possible attributes that can uniquely identify the rows in the given relation.
- A table can have many super keys.
- A super key may have additional attributes that are not needed for unique identity.

STUDENT Table

RollNo	Email	Phone	Name
101	a@gmail.com	9876543210	Ravi

102	b@gmail.com	9123456789	Anu

Possible Super Keys

- {RollNo}
- {Email}
- {Phone}
- {RollNo, Name}
- {Email, Phone}
- {RollNo, Email, Phone}

Candidate Key

- It is a minimal super key.
- It is called a minimal super key because we select a candidate key from a set of super keys such that we select a candidate key with the minimum attributes required to uniquely identify the table.
- Candidate keys are defined as attribute sets from which the primary key can be selected.

Example

STUDENT Table

RollNo	Email	Phone	Name
101	a@gmail.com	9876543210	Ravi
102	b@gmail.com	9123456789	Anu

Candidate Keys

- {RollNo}
- {Email}
- {Phone}

Alternate Key

- Out of all candidate keys, only one is selected as the primary key; the remaining keys are known as alternate keys.

Example
STUDENT Table

RollNo	Email	Phone	Name
101	a@gmail.com	9876543210	Ravi
102	b@gmail.com	9123456789	Anu

Candidate Keys

- {RollNo}

- {Email}
- {Phone}

Primary Key (chosen)

- {RollNo}

Alternate Keys

- {Email}
- {Phone}

Foreign Key

- Key used to link two tables together.
- Key to ensure referential integrity of the data.
- Foreign key references the primary key of another table.
- Foreign key can take only those values which are present in the primary key of the referenced relation.
- Foreign key can take the NULL value.
- There is no restriction on a foreign key to be unique.
- Referenced relation may also be called the **master table** or **primary table**.
- Referencing relation may also be called the **foreign table** or **detailed table**.

Example

STUDENT Table

StudentID (PK)	Name
1	Ravi
2	Anu

COURSE Table

CourseID (PK)	CourseName	StudentID (FK)
101	DBMS	1
102	OS	2

✔ StudentID in **COURSE** is a **foreign key**

✔ It references StudentID in **STUDENT**

Composite Key

- If one attribute is not enough to identify the tuple, then we need to identify more number of attributes.

- Such attributes are called a **composite key**.

Example

ENROLLMENT Table

StudentID	CourseID	Semester
1	101	Sem1
1	102	Sem1
2	101	Sem1

✔ StudentID alone → **not unique**

✔ CourseID alone → **not unique**

✔ {StudentID, CourseID} → **unique**

So, **{StudentID, CourseID}** is a **composite key**.

Participation Constraints
A **participation constraint** is applied on the entity participating in the relationship set.

There are two types of participation constraints:

1. **Total Participation**
2. **Partial Participation**

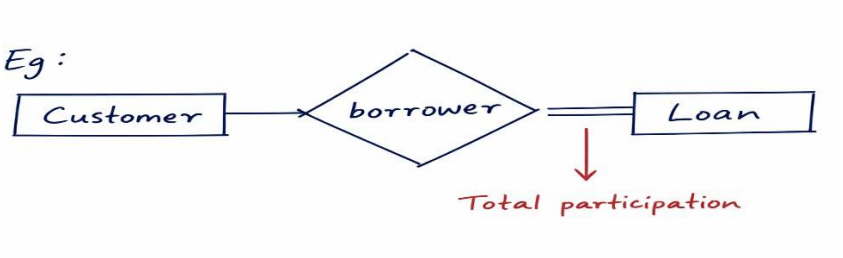
1) Total Participation

- If every entity in the entity set **E** participates in the relationship set **R**,
  then we say the participation of the entity set **E** with relationship set **R** is **total**.

- **Total participation** is represented by **double lines**.

  Eg: Participation of **Loan** in **Borrows** is **total**.

- Every **loan** must have a **customer** associated to it via **Borrows**.



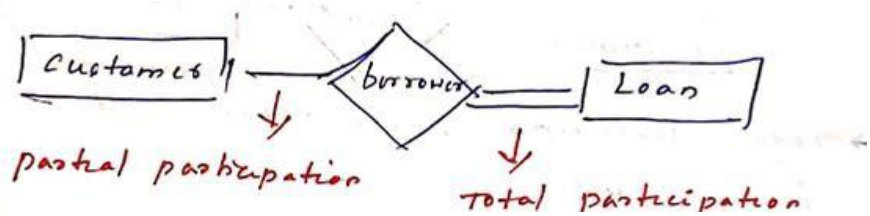
Every **department** should be managed by **at least one employee**.

2) Partial Participation

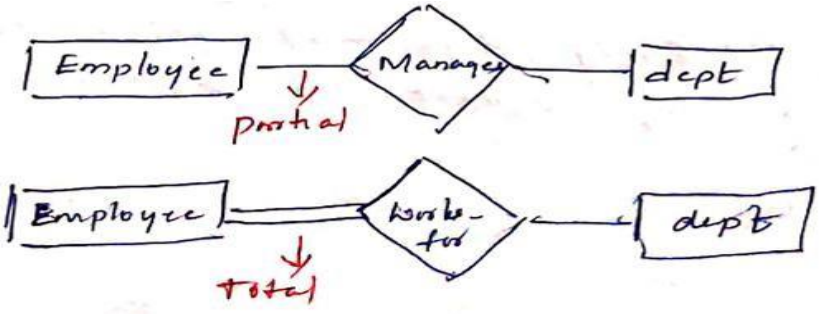
- If **only some entities** in **entity set E** participate in a **relationship set R**, then the participation of such **entity set E** with that **relationship set R** is **partial**.
- **Partial participation** is represented using **single lines**.

Eg:

- Participation of **Customer** entity set in **Borrows** relationship
- Participation of **Employee** entity set in **Manages** relationship



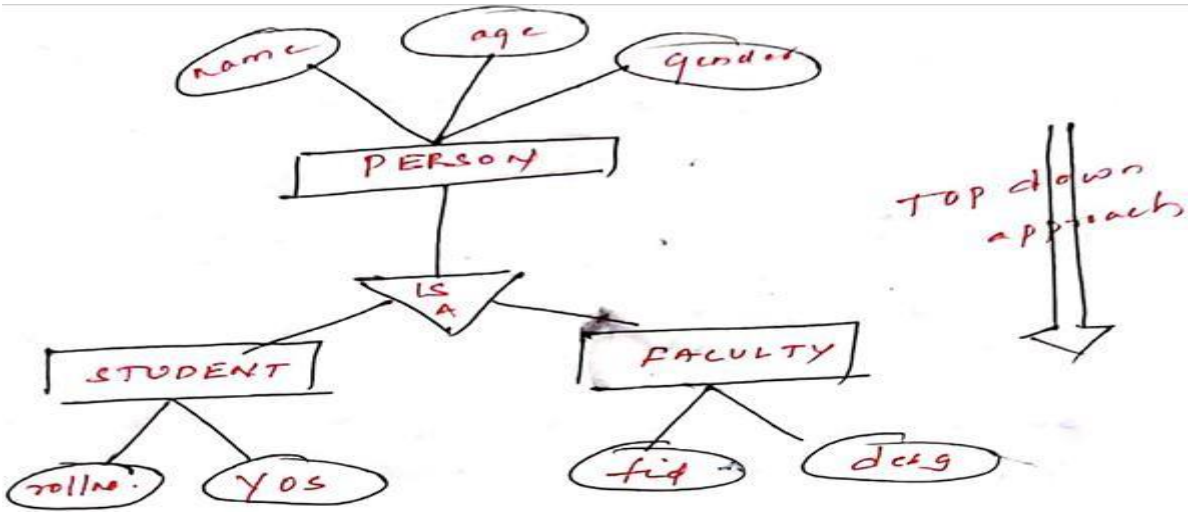
- Every **loan** is borrowed by **at least one customer**, so the participation of the **Loan** entity set with the **Borrows** relationship is **total participation**.
- Only **some customers** will borrow loans, so the participation of the **Customer** entity set in the **Borrows** relationship is **partial participation**.



Specialization

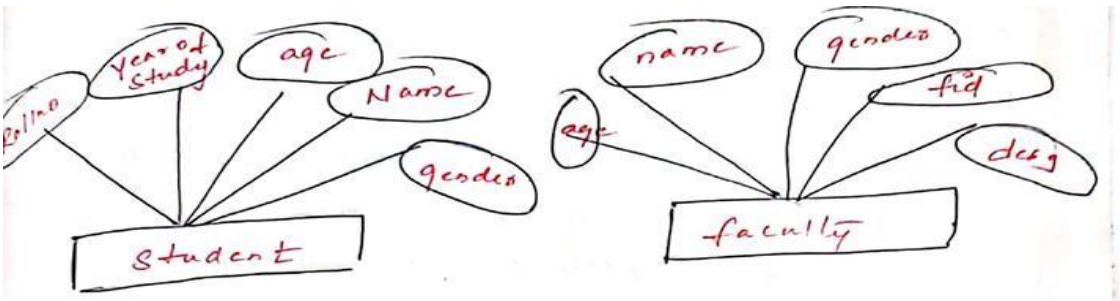
- Specialization is opposite to **Generalization**.
- Specialization is a process of identifying **sub-entities** of an entity set that share different characteristics.
- It is the process of dividing the **higher-level entity set** into **lower-level entity sets** based on specific characteristics.
- It is a **top-down approach**.

This **higher-level entity set** can be subdivided into **two lower-level entity sets** based on its **specific attributes**.



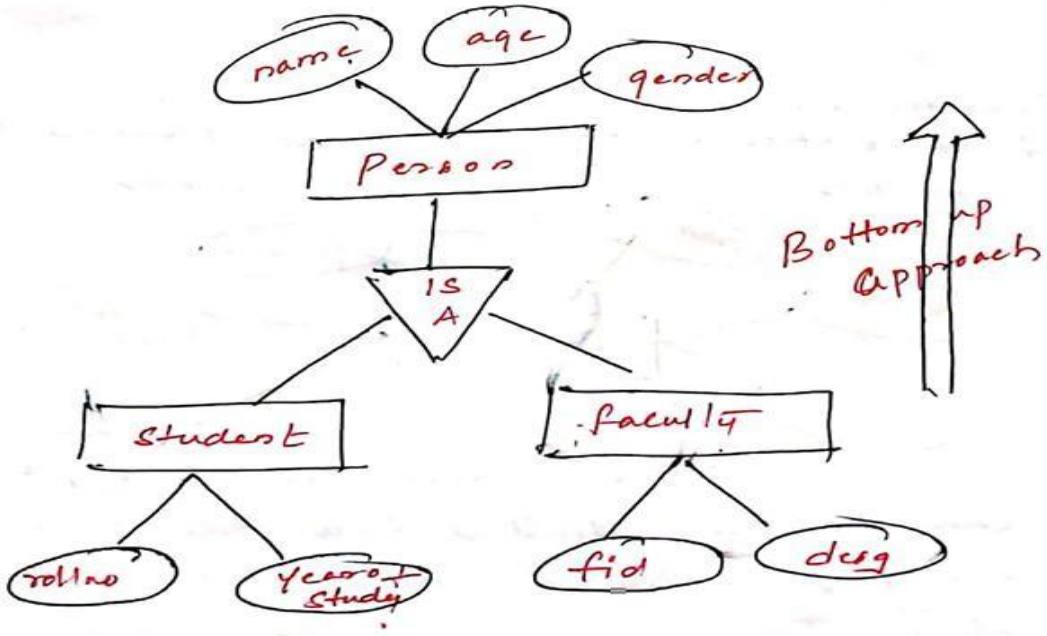
Generalization

- **Generalization** is the process of extracting **common properties** from **lower-level entity sets** and creating a **generalized (higher-level) entity set** from them.
- Generalization is a **bottom-up approach**, in which **two or more entity sets** are combined to form a **higher-level entity set**, provided they have **some attributes in common**.



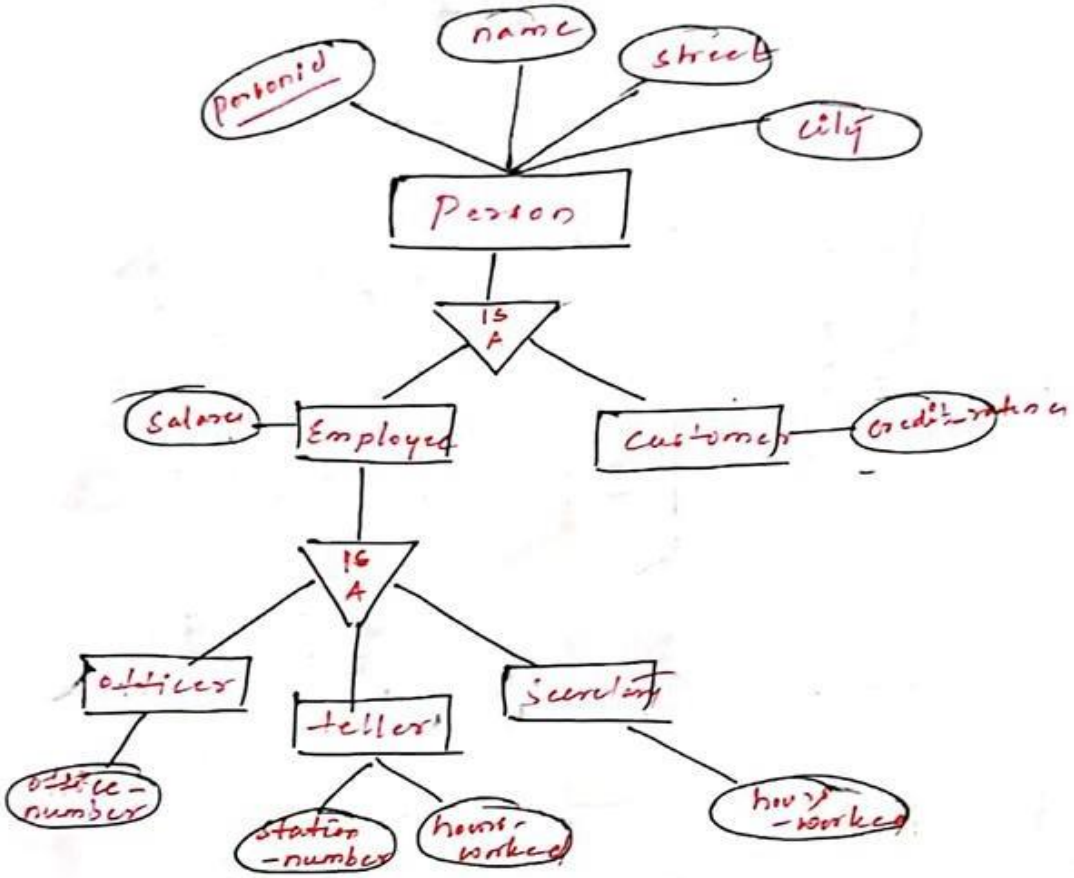
Prepared: V.Ragubathy

- When we follow a **bottom-up approach**, first we identify **two lower-level entity sets**.
- These two lower-level entity sets have **common attributes** such as **age, name, gender**, etc.
- Therefore, these two lower-level entity sets can be **generalized (or synthesized)** into a **higher-level entity set** called **Person**.



Attribute Inheritance

1. Lower-level entity sets are created by **specialization**.
2. Higher-level entity sets are created by **generalization**.
3. A common property of the higher-level and lower-level entities created by specialization and generalization is **attribute inheritance**.
4. The attributes of the **higher-level entity set** are said to be **inherited by the lower-level entity sets**.
5. Attribute inheritance allows **lower-level entities to inherit the attributes of higher-level entities**.



For instance, **Customer** and **Employee** inherit the attributes of **Person**.

Officer, **Teller**, and **Secretary** inherit all the attributes of **Employee** and **Person**.

- In a **hierarchy**, if a lower-level entity set is in **only one ISA relationship**, then the entity set has **single inheritance**.
- If a lower-level entity set is in **more than one ISA relationship**, then the entity set has **multiple inheritance**.
- The resulting structure is said to be a **lattice**.

TEXT BOOKS:

1. Database Management Systems, 3rd edition, Raghurama Krishnan, Johannes Gehrke, TMH (For Chapters 2, 3, 4)
2. Database System Concepts,5th edition, Silberschatz, Korth, Sudarsan,TMH (For Chapter 1 and Chapter

REFERENCE BOOKS:

1. Introduction to Database Systems, 8thedition, C J Date, Pearson.
2. Database Management System, 6th edition, RamezElmasri, Shamkant B. Navathe, Pearson
3. Database Principles Fundamentals of Design Implementation and Management, Corlos Coronel, Steven Morris, Peter Robb, Cengage Learning.

UNIT-II

Introduction to relational model:

The relational data model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper (Codd 1970), and it attracted immediate attention due to its simplicity and mathematical foundation.

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a flat file of records. It is called a flat file because each record has a simple linear or flat structure.

When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row. **Data Model** defines how the logical structure of a database is modeled.

Example: In STUDENT relation because each row represents facts about a particular student entity. The column names Name, Student_ number, Class, and Major specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

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

concepts of domain, attribute, tuple, relation:

Domain:

A **domain** D is a set of atomic values. By **atomic** we mean that each value in the domain is invisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the

domain are drawn. It is also useful to specify the name for the domain, to help in interpreting its values.

Some examples of domains follow:

- Usa_phone_numbers: The set of ten-difgit phone numbers valid in United States.
- Social_security_numbers: The set of valid nine-digit social security numbers.
- Names: The set of character strings that represents the names of persons.
- Employee_ages: Possible ages of employees in a company; each must be an integer value between 15 and 80.

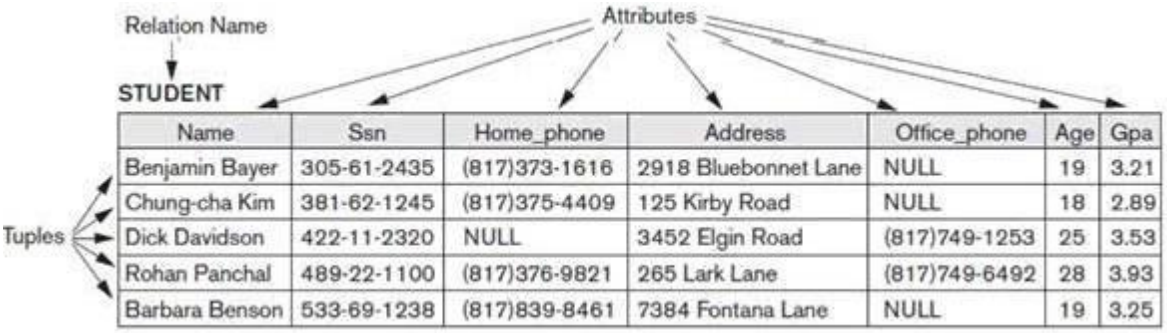
Tuple:

Mapping from attributes to values drawn from the respective domains of those attributes. Tuples are intended to describe some entity (or relationship between entities) in the miniworld **Example:** a tuple for a PERSON entity might be

{ Name □□"smith", Gender□Male, Age □25 }

Relation:

A named set of tuples all of the same form i.e., having the same set of attributes.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Importance of null values:
In a Database Management System (DBMS), a null value represents missing, unknown, or inapplicable data. Unlike zero or an empty string, a null value explicitly indicates that no value is assigned to a particular field.
constraints (Domain, Key constraints, integrity constraints) and their importance:

Domain Constraints Definition:
Domain constraints define the **permissible values** for a column (attribute) in a table. Each column has a data type that restricts the values it can store.
Example:
CREATE TABLE Employees (emp_id INT PRIMARY KEY, name VARCHAR(50),
age INT CHECK (age > 18 AND age < 65), salary DECIMAL(10,2)
);

Here, the age column must be **greater than 18 and less than 65**, ensuring that only valid employee ages are stored.

Importance of Domain Constraints:
 Prevents **invalid data** entry (e.g., entering text in a numeric field).

 Ensures **data accuracy** by restricting values to a valid range.

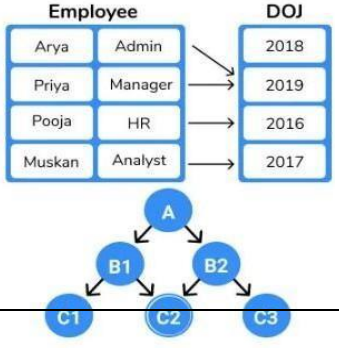
 Reduces **data inconsistencies** and errors.

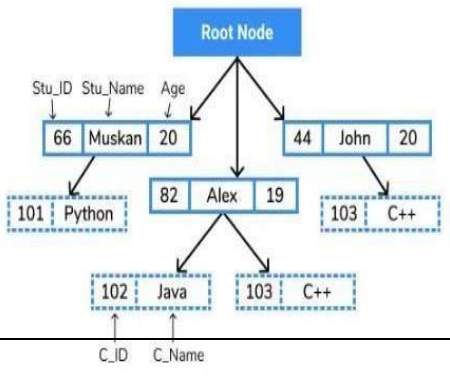
Key Constraints Definition:

Key constraints ensure **uniqueness** and **uniquely identify** records in a table. The main types of key constraints are:

Name	Street	City	Number
A	Maple	Queens	900
B	Norris	Bronx	556
C	Willis	Bronx	647
H	Schilly	Brooklyn	801

Number	Balance
900	55
556	1,100,000
647	1,004,26
801	1,105,33





ıy

- Relational Model was proposed by **E. F. Codd** to model data in the form of **relations (tables)**.
- After designing the **conceptual model** of a database using an **ER diagram**, we need to convert the conceptual model into the **logical model (Relational Model)**, which can be implemented using any **RDBMS language** like **Oracle, MySQL**, etc.
- The **Relational Model** is the most popular used model for representing **data and its relationships**.
- In the **Relational Model**, both **data and relationships** are represented in the form of **relations (tables)**.

Degree of Employee Relation = **3** (Number of attributes)

Cardinality = **4** (Number of tuples)

Domain Constraint

- **Domain integrity** means the definition of a **valid set of values** for an attribute.
- It involves **stating the possible values** that an attribute can take **at the time of schema definition**, such as:
    - Data type
    - Size (length)

EXAMPLE:

```
CREATE TABLE Emp (
   Eno NUMBER(2),
   Ename VARCHAR2(10)
);
```

- NUMBER(2) → **Domain constraint of Eno**
- VARCHAR2(10) → **Domain constraint of Ename**
- Allows **all possible 1-digit and 2-digit numbers**.

Examples of Domains

- **Age** → Integer values between 18 and 60
- **Salary** → Positive numbers only
- **Gender** → {Male, Female, Other}
- **PhoneNo** → Exactly 10 digits

Domain is Important

1. **Ensures data validity**
   → Prevents incorrect data entry.
2. **Maintains data consistency**
   → Same type of data stored uniformly.
3. **Improves data integrity**
   → Invalid values are rejected.
4. **Simplifies error checking**
   → Rules are enforced automatically.

Domain Constraint (Rule)

A **domain constraint** ensures that attribute values always come from the defined domain.

Example:
If **Age domain = 18–60**, inserting Age = 10is **not allowed**.

Attribute

An **attribute** is a **property or characteristic** of an entity.
In a relational database, an attribute represents a **column** in a table.

An attribute is a named column in a relation that describes a specific property of an entity.

Example

Relation (Table):

EMPLOYEE (EmpID, Name, Salary, Department)

- EmpID → Attribute
- Name → Attribute
- Salary → Attribute
- Department → Attribute

Each attribute has a **domain** that defines its possible values.

Types of Attributes

1. **Simple (Atomic) Attribute**
   - Cannot be divided further
   - Example: Age, RollNo
2. **Composite Attribute**
   - Can be divided into sub-parts
   - Example: Address → Street, City, PIN
3. **Single-Valued Attribute**
   - Has only one value per entity

- Example: DateOfBirth

4. **Multi-Valued Attribute**
    - Can have multiple values
    - Example: PhoneNumbers
5. **Derived Attribute**
    - Value is calculated from other attributes
    - Example: Age (derived from DateOfBirth)

Attribute Domain

- Each attribute is associated with a **domain**
- Domain specifies:
    - Data type
    - Range
    - Format

Example:

Name VARCHAR(50)

Salary INT

Importance of Attributes

1. Describe the **structure of data**
2. Help identify **entity properties**
3. Ensure **data consistency** using domains
4. Form the **degree** of a relation

Relation importance of null values

What is a Relation?

A **relation** is a **table** that consists of **rows (tuples)** and **columns (attributes)**.

Definition

A relation is a collection of tuples sharing the same attributes.

Example: EMPLOYEE Relation

EmpID	Name	Salary
1	A	50000
2	B	60000

- Table name → EMPLOYEE

- Columns → Attributes
- Rows → Tuples

Properties of a Relation

1. Relation name must be **unique**
2. Attribute names must be **unique**
3. Order of rows and columns **does not matter**
4. Each cell contains **atomic (single) values**
5. No duplicate tuples allowed

Importance of Relation

1. Organizes data in a **structured tabular form**
2. Easy to **store, retrieve, and update** data
3. Represents **real-world entities**

NULL Values and Their Importance

What is a NULL Value?

A **NULL value** represents:

- Missing data
- Unknown value
- Not applicable value

NOTE: NULL is **not** zero and **not** an empty string.

Examples of NULL Values

EmpID	Name	Phone
1	A	NULL

- Phone number is unknown or not provided

Importance of NULL Values

1. Represents **incomplete information**
2. Reflects **real-world situations**
3. Allows optional attributes

Constraints in Relational Model

Constraints are **rules** applied to database tables to ensure **accuracy, consistency, and integrity** of data.

1. **Domain Constraints**
2. **Key Constraints**

3. Integrity Constraints

1. Domain Constraints

Domain constraints specify that the **values of an attribute must belong to a predefined domain** (data type, range, or format).

Example

- Age must be between 1 and 100
- Gender must be Male or Female

Age INT CHECK (Age > 0),

Gender VARCHAR(6) CHECK (Gender IN ('Male','Female'))

Importance

1. Prevents invalid data entry
2. Maintains data consistency
3. Ensures correct data type and range

2. Key Constraints

Key constraints ensure that **each tuple in a relation is uniquely identifiable**.

Types of Keys

1. **Super Key**
    - A set of one or more attributes that uniquely identifies a tuple
    - Example: (EmpID), (EmpID, Name)
2. **Candidate Key**
    - Minimal super key
    - Example: EmpID
3. **Primary Key**
    - Selected candidate key
    - Cannot be NULL or duplicate
4. **Foreign Key**
    - Attribute that references a primary key of another table

Example

EmpID INT **PRIMARY KEY**

DeptID INT REFERENCES Department(DeptID)

Importance

1. Ensures uniqueness of records

2. Avoids duplicate data
3. Maintains relationships between tables

3. Integrity Constraints

Integrity constraints ensure **correctness and validity of data relationships**.

Types of Integrity Constraints

a) Entity Integrity Constraint

- Primary key **cannot be NULL**
- Ensures each tuple is uniquely identifiable

b) Referential Integrity Constraint

- Foreign key must:
    - Match a primary key in referenced table OR
    - Be NULL

Example

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

Importance

1. Maintains relationship consistency
2. Prevents invalid references
3. Ensures reliable and accurate data

Relational Algebra

- Relational Algebra is a **query language**.
- Query language is a language in which users request information from the database.
- SQL is a query language.

Two types of Query Languages are:

1. **Procedural Query Language**
2. **Non-Procedural (Declarative) Query Language**

1. Procedural Query Language

- In a procedural query language, the user instructs the system to perform a **sequence of operations** to produce the desired result.
- The user tells **what data to be retrieved** from the database **and how to retrieve it**.

2. Non-Procedural (Declarative) Query Language

- In a non-procedural query language, the user instructs the system to produce the

desired result **without telling the step-by-step procedure**.

- The user tells **what data is to be retrieved**, but **does not tell how to retrieve it**.

Example:

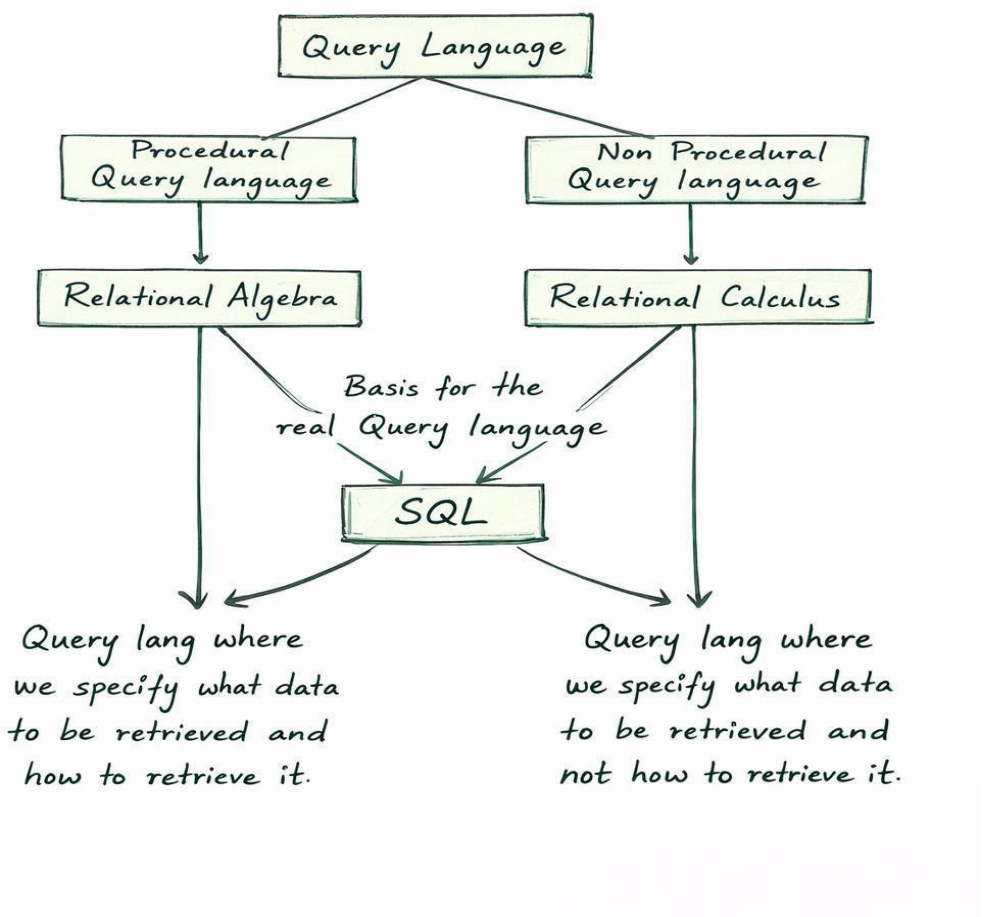
- If we ask our daughter or son to **bring a cup of tea**, then it comes under **Non-Procedural Query Language** (we ask them to get a cup of tea).
- If we ask them to get a cup of tea **and also tell them step by step procedure to make tea**, then it comes under **Procedural Query Language**.

Two pure or basic Query Languages are:

1. **Relational Algebra** (Procedural Query Language)
2. **Relational Calculus** (Non-Procedural Query Language)

Types of Relational Calculus:

- **Tuple Relational Calculus**
- **Domain Relational Calculus**



Relational Algebra

- It is a **procedural query language**, in which the user should instruct the system a **series of operations** to get the desired result.
- Hence, the user must tell **what data** to be retrieved and **how** to retrieve it.
- It takes a **relation as an input** and generates a **relation as an output**.
- It uses **operators** to perform queries.
- An operator can be **unary** or **binary**.

Basic operations	Additional operations
1. Selection (σ)	1. Natural Join ($\bowtie$)
2. Projection (π)	2. Left, Right, Full outer Join
3. Union ($\cup$)	3. Set Intersection ($\cap$)
4. Set Difference ($-$)	4. Division ($\div$)
5. Cartesian Product ($\times$)	5. Assignment ($\leftarrow$)
6. Rename (ρ)	

Sample Relations

Consider the following relations:

EMPLOYEE(EmpID, EmpName, DeptID, Salary)

EmpID	EmpName	DeptID	Salary
1	Sai	10	45000
2	Anu	20	38000
3	Rahul	10	50000
4	Meena	30	42000

DEPARTMENT(DeptID, DeptName)

DeptID	DeptName
10	HR
20	IT
30	Finance

Fundamental Operations of Relational Algebra

1. Selection (σ)

Selection is a unary relational algebra operation that selects tuples from a relation that satisfy a specified condition.

It is used to filter rows based on comparison, arithmetic, or logical expressions.

Notation

σ(Relation)

Selection scans each tuple of the relation and checks whether the tuple satisfies the condition. Only those tuples that satisfy the condition are included in the output relation.

Example

Select employees whose salary is greater than 40000.

σ Salary > 40000 (EMPLOYEE)

Result

EmpID	EmpName	DeptID	Salary
1	Sai	10	45000
3	Rahul	10	50000
4	Meena	30	42000

Properties

1. Number of attributes remains same.
2. Number of tuples may decrease.
3. Condition may use AND, OR, NOT.
4. Equivalent to SQL WHERE clause.

2. Projection (π)

Projection is a unary operation that selects specific attributes (columns) from a relation.

It eliminates duplicate tuples automatically.

Notation

π attribute-list (Relation)

Projection creates a new relation containing only specified attributes. Duplicate rows are removed because relational algebra follows set semantics.

Example

Display employee names and salaries.

π EmpName, Salary (EMPLOYEE)

Result

EmpName	Salary
Sai	45000
Anu	38000
Rahul	50000
Meena	42000

Properties

1. Reduces number of attributes.
2. Removes duplicate tuples.
3. Changes relation structure.
4. Equivalent to SQL SELECT column-list.

3. Union (∪)

Union is a binary operation that combines tuples from two union-compatible relations.

Union Compatibility Conditions

1. Same number of attributes.
2. Corresponding attributes must have same domain.
3. Attribute order must match.

Example

Let R = Employees in Dept 10
Let S = Employees in Dept 30

R ∪

Result

EmpID	EmpName	DeptID	Salary
1	Sai	10	45000
3	Rahul	10	50000
4	Meena	30	42000

Properties

1. Removes duplicates.
2. Requires union compatibility.
3. Equivalent to SQL UNION.

4. Set Difference (−)

Set Difference returns tuples that are present in the first relation but not in the second.

Example

Employees in Dept 10 − Employees with Salary > 48000

Result

EmpID	EmpName	DeptID	Salary
1	Sai	10	45000

Properties

1. Requires union compatibility.
2. Order of operands matters.
3. Equivalent to SQL MINUS.

5. Cartesian Product (×)

Cartesian product combines every tuple of one relation with every tuple of another relation.

If R has m tuples and S has n tuples, then R × S has m × n tuples.

Example

EMPLOYEE × DEPARTMENT

Number of tuples = 4 × 3 = 12

Cartesian product is rarely used alone. It is mainly used as an intermediate step in join operations.

6. Join (⋈)

Join operation combines related tuples from two relations based on a specified condition.

Join = Selection applied to Cartesian Product.

$$R \bowtie S = \sigma_{condition}(R \times S)$$

Example

EMPLOYEE ⋈ DEPARTMENT
Condition: EMPLOYEE.DeptID = DEPARTMENT.DeptID

Result

EmpID	EmpName	DeptID	Salary	DeptName
1	Sai	10	45000	HR
2	Anu	20	38000	IT
3	Rahul	10	50000	HR
4	Meena	30	42000	Finance

7. Intersection (∩)

Intersection returns tuples that are common to both relations.

$$R \cap S$$

Example

Employees in Dept 10 ∩ Employees with Salary > 48000

Result:

EmpID	EmpName	DeptID	Salary
3	Rahul	10	50000

8. Rename (ρ)

Rename operation changes the name of a relation or its attributes.

Used when performing self-joins.

Example:

ρ EMP (EMPLOYEE)

9. Division (÷)

Division is used to find tuples in one relation that are associated with all tuples in another relation.

It supports "for all" queries.

Example:

Find employees who worked in all departments.

Relational Calculus

1. Relational Calculus is a non-procedural or declarative language.
2. Relational Calculus tells **what to do** but never explains **how to do**.
3. When applied to database, it comes in 2 flavors:

1) Tuple Relational Calculus (TRC)

- Proposed by Codd in the year 1972
- Works on tuple

2) Domain Relational Calculus (DRC)

- Proposed by Lacroix and Pirotte in 1977
- Works on domain of attribute

4. Calculus has variables, constants, comparison operator, logical connectives and quantifiers.
5. Tuple Relational Calculus is a non-procedural query language.
6. TRC is used for selecting the tuples in a relation that satisfy the given condition.

A query in TRC is expressed as:

$$\{ t \mid P(t) \}$$

Where:

- t → denotes resulting tuple
- P(t) → denotes predicate used to fetch tuple t

Query returns set of all tuples t such that predicate P is true for t.

Procedural vs Non-Procedural

Relational Algebra	Relational Calculus
Procedural	Non-Procedural
Specifies steps	Specifies condition
Tells HOW	Tells WHAT

Example Difference

Relational Algebra

π emp_name (σ salary > 50000 (Employee))

Relational Calculus

{ t | t ∈ Employee AND t.salary > 50000 }

Both give same result — but approach is different.

Types of Relational Calculus

There are two types:

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

1. Tuple Relational Calculus (TRC)

Tuple Relational Calculus uses **tuple variables** to describe queries.

General Form

[
{ t | P(t) }
]

Where:

- t → tuple variable
- P(t) → condition (predicate)

Sample Relation

Employee

emp_id	emp_name	salary	dept_id
1	Sai Kiran	45678.50	10
2	Anu Priya	38900.75	20
3	Rahul Dev	50234.00	10

Example

Find employees with salary > 40000

TRC Expression

[
{ t | t ∈ Employee AND t.salary > 40000 }
]

Output

emp_id	emp_name	salary	dept_id
1	Sai Kiran	45678.50	10
3	Rahul Dev	50234.00	10

Example 2

Find employee names working in dept 10

[
{ t.emp_name | t ∈ Employee AND t.dept_id = 10 }
]

Output

emp_name
Sai Kiran
Rahul Dev

2. Domain Relational Calculus (DRC)

Domain Relational Calculus uses **domain variables** (attribute-level variables) instead of whole tuples.

General Form

[
{ <x1, x2, ..., xn> | P(x1, x2, ..., xn) }
]

Example

Find employee names with salary > 40000

[
{ | ∃ id, s, d (Employee(id, n, s, d) AND s > 40000) }
]

Where:

- id → emp_id
- n → emp_name
- s → salary
- d → dept_id

Important Symbols Used

Symbol	Meaning
∈	belongs to
∧	AND
∨	OR
¬	NOT
∃	There exists
∀	For all

1. Existential Quantifier (∃)

Means "There exists"

Example

Find employees who work in some department

[
{ t | ∃ d (Department(d) AND t.dept_id = d.dept_id) }
]

2. Universal Quantifier (∀)

Means "For all"

Example:
Find employees who work in all projects

[
{ t | ∀ p (Project(p) → Works(t,p)) }
]

3. Safe vs Unsafe Expressions

- **Safe Expression**

  Produces finite result.

- **Unsafe Expression**

  May produce infinite results.

  Example of unsafe:

[
{ t | ¬ (t ∈ Employee) }
]

This tries to return all tuples not in Employee (infinite).

4. **Relationship with SQL**

Relational Calculus is theoretical foundation of SQL.

Example:

SQL

SELECT emp_name

FROM Employee

WHERE salary > 40000;

TRC Equivalent

[
{ t.emp_name | t ∈ Employee AND t.salary > 40000 }
]

SIMPLE DATABASE SCHEMA

A **database schema** is the overall logical design of a database. It is a collection of relation schemas that describe:

- The structure of the data
- The names of relations
- The attributes of each relation
- The domains of attributes
- Integrity constraints

A schema is defined using the **Data Definition Language (DDL)**.

Important Concepts

Relation Schema

A relation schema consists of:

[
R(A_1, A_2, A_3, ..., A_n)
]

Where:

- R → relation name
- A_1, A_2, ... → attributes

Example:

[
Student(student_id, name, dept_name, tot_cred)
]

DATABASE LANGUAGE – TYPES

1. Data Definition Language (DDL)

DDL is used to define and modify the structure of database objects.

It includes:

- Creating tables
- Defining constraints
- Modifying schema
- Dropping relations

DDL statements are compiled and stored in the data dictionary.

2. Data Manipulation Language (DML)

DML is used to retrieve, insert, delete, and modify data stored in relations.

Two types of DML:

1. Procedural DML – User specifies what data and how to retrieve
2. Non-Procedural DML – User specifies what data is required (SQL SELECT)

3. Data Control Language (DCL)

Controls access to database.

- GRANT
- REVOKE

4. Transaction Control Language (TCL)

Manages transactions.

- COMMIT
- ROLLBACK

TABLE DEFINITIONS (DDL OPERATIONS)

1. CREATE TABLE

The CREATE TABLE statement is used to define a new relation schema by specifying:

- Relation name
- Attributes
- Data types
- Integrity constraints

Syntax

```
CREATE TABLE table_name (
   attribute_name data_type [constraint],
   attribute_name data_type [constraint],
   ...
);
```

Explanation of Components

Data Types

Common data types:

- INT / INTEGER

- VARCHAR(n)
- CHAR(n)
- DATE
- DECIMAL(p,s)

Constraints

Constraints ensure data integrity.

Examples:

- PRIMARY KEY
- FOREIGN KEY
- UNIQUE
- NOT NULL
- CHECK

Example

```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50) NOT NULL,
   salary DECIMAL(10,2) CHECK (salary > 0),
   dept_name VARCHAR(30)
);
```

Conceptual Output

An empty table is created:

emp_id	emp_name	salary	dept_name
—	—	—	—

2. ALTER TABLE

ALTER TABLE modifies an existing table definition without deleting data.

Operations include:

- Add column
- Drop column
- Modify column

- Add constraint

Add a column:

ALTER TABLE Employee

ADD join_date DATE;

3. DROP TABLE

Removes entire relation and its data permanently.

DROP TABLE Employee;

DATA MANIPULATION LANGUAGE (DML OPERATIONS)

1. INSERT

INSERT adds new tuples into a relation.

It ensures inserted values satisfy domain constraints and integrity constraints.

Syntax

INSERT INTO table_name

VALUES (value1, value2, ...);

Example

INSERT INTO Employee

VALUES (1, 'Sai Kiran', 45000, 'CSE');

Result

emp_id	emp_name	salary	dept_name
1	Sai Kiran	45000	CSE

2 DELETE

DELETE removes tuples that satisfy a specified predicate.

Syntax

DELETE FROM table_name

WHERE condition;

Example

DELETE FROM Employee

WHERE emp_id = 1;

3. UPDATE

UPDATE modifies attribute values of tuples satisfying a condition.

Syntax

UPDATE table_name

SET attribute = value

WHERE condition;

Example

UPDATE Employee

SET salary = 50000

WHERE emp_id = 1;

TEXT BOOKS:

1. Database System Concepts, Silberschatz, Korth, Sudarsan,TMH.

2. Data base Management Systems, Raghurama Krishnan, Johannes Gehrke, TATA McGrawHill .

REFERENCE BOOKS:

1. Fundamentals of Database Systems, Elmasri Navathe Pearson Education.

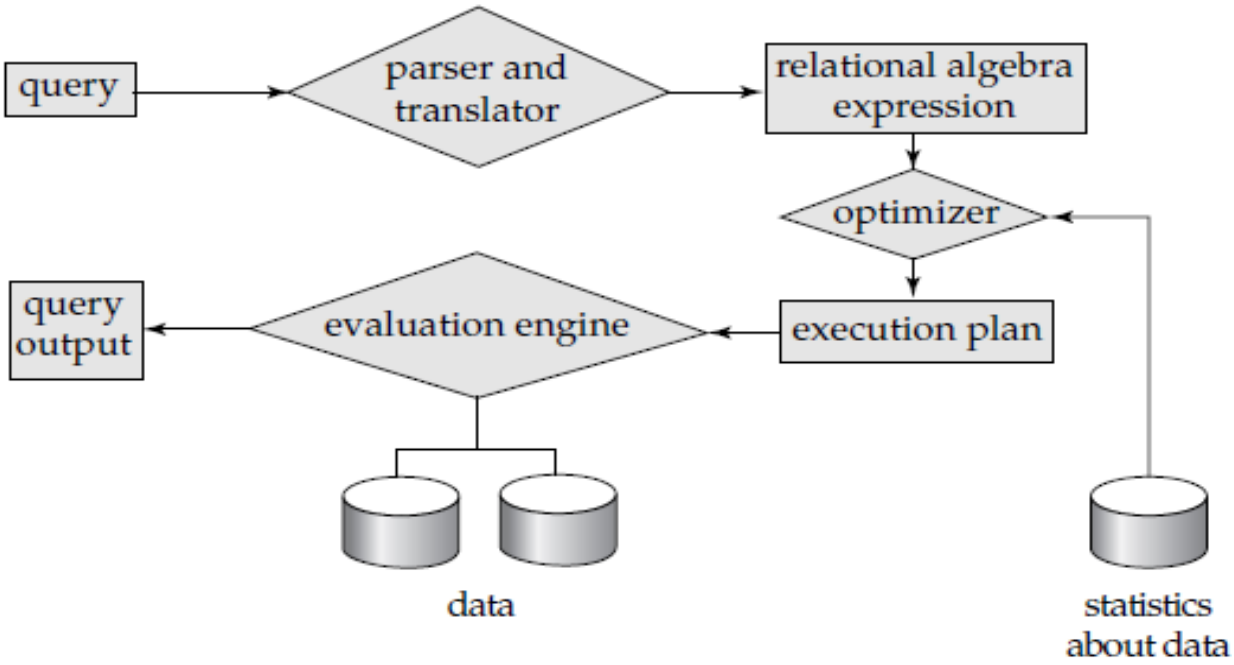
REFERENCE WEBSITE:

1. **https://www.w3schools.in/sql/database-concepts**
2. **https://www.javatpoint.com/dbms-tutorial**
3. **https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/**

UNIT-III

Structured Query Language (SQL)

- The SQL programming language was first developed in the 1970s by IBM researchers Raymond Boyce and Donald Chamberlin.
- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Oracle, MS Access and SQL Server use SQL as their standard database language.



- Efficient
- Easy to learn and use
- With SQL, you can define, retrieve, and manipulate data in the tables

CHARACTERISTICS:

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.

- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

- There are the following advantages of SQL:

1.High speed

- Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

2.No coding needed

- In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

3.Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

4.Portability

SQL can be used in laptop, PCs, server and even some mobile phones

5.Interactive language

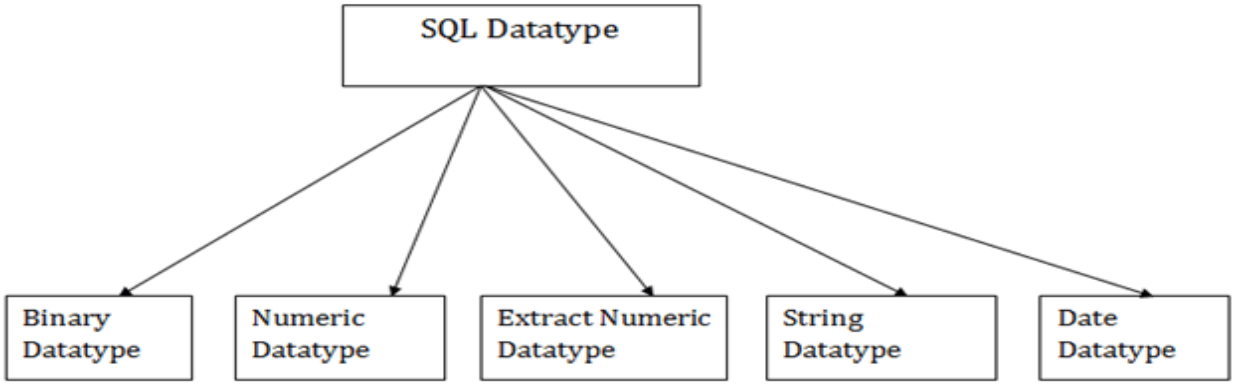
SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

6.Multiple data view

Using the SQL language, the users can make different views of the database structure.

SQL Datatype

- SQL Datatype is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.



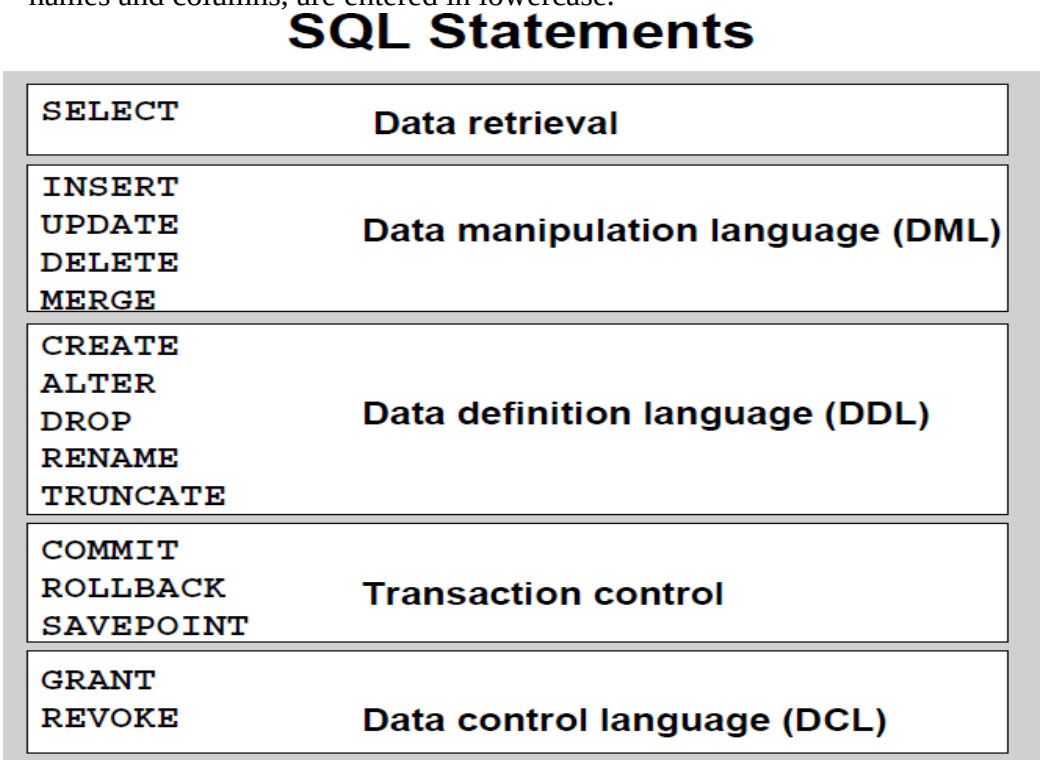
SQL Standards:

Oracle SQL complies with industry-accepted standards. . Industry-accepted committees are the American National Standards Institute (ANSI). And the International Standards Organization (ISO) . Both ANSI and ISO have accepted SQL as the standard language for relational databases.

Writing SQL Statements

Using the following simple rules and guidelines, you can construct valid statements that are both easy to read and easy to edit:

- SQL statements are not case sensitive.
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns, are entered in lowercase.



Data types:

When you create a table or cluster, you must specify a data type for each of its columns. When you create a procedure These data types define the domain of values that each column can contain or each argument can have.

CHAR(size) :-

Fixed-length character data of length size bytes. Maximum size is 2000 bytes. Default

and minimum size is 1 byte.

VARCHAR2(size) :-

Variable-length character string having maximum length size bytes or characters. Maximum size is 4000 bytes, and minimum is 1 byte or 1 character. You must specify size for VARCHAR2.

NCHAR(size):-

Fixed-length character data of length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 2000 bytes. Default and minimum size is 1 character or 1 byte, depending on the character set.

NVARCHAR2(size) :

Variable-length character string having maximum length size characters or bytes, depending on the choice of national character set. Maximum size is determined by the number of bytes required to store each character, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2.

NUMBER(p,s) :

Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127

LONG :

Character data of variable length up to 2 gigabytes, or 231 -1 bytes.

DATE :

Allows date & time but Time is optional if not entered by user then oracle inserts 12:00AM. Valid date range from January 1, 4712 BC to December 31, 9999 AD. a Date field occupies 7 bytes of memory

RAW(size) :

Raw binary data of length size bytes. Maximum size is 2000 bytes. You must specify size for a RAW value.

LONG RAW :

Raw binary data of variable length up to 2 gigabytes.

ROWID :

Hexadecimal string representing the unique address of a row in its table. This datatype is primarily for values returned by the ROWID pseudocolumn.

UROWID [(size)] :

Hexadecimal string representing the logical address of a row of an index-organized table. The optional size is the size of a column of type UROWID. The maximum size and default is 4000 bytes.

CLOB:

A character large object containing single-byte characters. Both fixed-width and variable-width character sets are supported, both using the CHAR database character set. Maximum size is 4 gigabytes.

NCLOB :

A character large object containing unicode characters. Both fixed-width and variable-width character sets are supported, both using the NCHAR database character set. Maximum size is 4 gigabytes. Stores national character set data.

BLOB :

A binary large object. Maximum size is 4 gigabytes.

BFILE :

Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes.

BINARY_FLOAT :

32-bit single precision floating point number datatype. Binary float equires 5 bytes including a length byte.

BINARY_DOUBLE:

64-bit double precision floating point number datatype. Binary double requires 9 bytes including a length byte.

Basic SQL Querying (SELECT & PROJECT) using WHERE Clause with Arithmetic and Logical Operations

1) SELECT Statement

The **SELECT** statement is used to retrieve data from one or more tables in a database.

Syntax:

SELECT column_list

FROM table_name;

Example:

SELECT * FROM Students;

Output:

roll_no	name	marks	age
1	Ravi	85	20
2	Sneha	72	19
3	Arjun	90	21
4	Meena	60	18
5	Kiran	88	20

2) Projection

Projection means selecting specific columns from a table instead of all columns.

Example:

SELECT name, marks FROM Students;

Output:

name	marks
Ravi	85
Sneha	72
Arjun	90
Meena	60

name	marks
Kiran	88

3) WHERE Clause

The **WHERE clause** is used to filter records based on a specified condition.

Syntax:

SELECT column_list

FROM table_name

WHERE condition;

a) Arithmetic Operations in WHERE

Arithmetic operators perform mathematical calculations inside SQL queries.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division

Example:

SELECT name, marks

FROM Students

WHERE marks + 5 >= 90;

Output:

name	marks
Ravi	85
Arjun	90
Kiran	88

b) Comparison Operators

Used to compare column values in the WHERE clause.

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal

Example:

SELECT * FROM Students

WHERE marks > 80;

Output:

roll_no	name	marks	age
1	Ravi	85	20
3	Arjun	90	21
5	Kiran	88	20

c) Logical Operators in WHERE

Logical operators combine multiple conditions in a query.

Operator	Meaning
AND	Both conditions must be true
OR	At least one condition must be true
NOT	Reverses the condition

AND Example

SELECT * FROM Students

WHERE marks > 75 AND age < 21;

Output:

roll_no	name	marks	age
1	Ravi	85	20
5	Kiran	88	20

OR Example

SELECT * FROM Students

WHERE marks > 85 **OR** age < 19;

Output:

roll_no	name	marks	age
3	Arjun	90	21
4	Meena	60	18
5	Kiran	88	20

NOT Example

SELECT * FROM Students

WHERE **NOT** marks < 70;

Output:

roll_no	name	marks	age
1	Ravi	85	20
2	Sneha	72	19
3	Arjun	90	21
5	Kiran	88	20

4) Combining Arithmetic + Logical Operators

Example

SELECT name, marks, age

FROM Students

WHERE (marks + 5) >= 90 AND age >= 20;

Output:

name	marks	age
Ravi	85	20
Arjun	90	21
Kiran	88	20

Special Filtering Operators

BETWEEN

Selects values within a given range.

SELECT * FROM Students **WHERE** marks BETWEEN 70 AND 85;

Output:

roll_no	name	marks	age
1	Ravi	85	20
2	Sneha	72	19

IN

Matches any value in a list.

SELECT * FROM Students **WHERE** age IN (18, 21);

Output:

roll_no	name	mar ks	age
3	Arjun	90	21
4	Meena	60	18

LIKE

Used for pattern matching.

SELECT * FROM Students **WHERE** name **LIKE** 'K%';

Output:

roll_no	name	marks	age
5	Kiran	88	20

Summary

Term	Definition
SELECT	Retrieves data from table
Projection	Selecting specific columns
WHERE	Filters rows
Arithmetic Ops	Perform calculations in query
Logical Ops	Combine conditions (AND, OR, NOT)
BETWEEN	Range filter
IN	Multiple value match
LIKE	Pattern matching

SQL functions (Date and Time, Numeric, String conversion)

Step 1: Create Sample Table

```
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    salary DECIMAL(10,2),
    join_date DATE
);
```

Insert Data

```
INSERT INTO Employees VALUES
(1, 'Sai Kiran', 45678.50, '2022-01-15'),
(2, 'Anu Priya', 38900.75, '2023-06-20'),
(3, 'Rahul Dev', 50234.00, '2021-11-05');
```

Table Data

emp_id	emp_name	salary	join_date
1	Sai Kiran	45678.50	2022-01-15
2	Anu Priya	38900.75	2023-06-20
3	Rahul Dev	50234.00	2021-11-05

NOTE: **AS is a keyword used to assign an alias (temporary name) to a column or table in a query.**

DATE & TIME FUNCTIONS

1) YEAR(join_date)

Extracts the year part from a date.

```
SELECT emp_name, YEAR(join_date) AS joining_year
FROM Employees;
```

Output:

emp_name	joining_year
Sai Kiran	2022
Anu Priya	2023
Rahul Dev	2021

2) MONTH(join_date)

Extracts the month from a date.

SELECT emp_name, **MONTH**(join_date) AS joining_month

FROM Employees;

Output:

emp_name	joining_month
Sai Kiran	1
Anu Priya	6
Rahul Dev	11

3) DATEDIFF(CURDATE(), join_date)

Returns number of days between today and joining date.

SELECT emp_name, **DATEDIFF**(CURDATE(), join_date) AS days_worked

FROM Employees;

Sample Output (depends on current date):

emp_name	days_worked
Sai Kiran	1490
Anu Priya	968
Rahul Dev	1550

4) DATE_ADD(join_date, INTERVAL 1 YEAR)

Adds 1 year to joining date.

SELECT emp_name, DATE_**ADD**(join_date, INTERVAL 1 YEAR) AS first_anniversary

FROM Employees;

Output:

emp_name	first_anniversary
Sai Kiran	2023-01-15
Anu Priya	2024-06-20
Rahul Dev	2022-11-05

NUMERIC FUNCTIONS

1) ROUND(salary, 0)

Rounds salary to nearest whole number.

SELECT emp_name, **ROUND**(salary,0) AS rounded_salary

FROM Employees;

Output:

emp_name	rounded_salary
Sai Kiran	45679
Anu Priya	38901
Rahul Dev	50234

2) ABS(-salary)

Converts negative value to positive.

ABS() is a **Numeric function** that returns the **absolute value** of a number.

SELECT emp_name, **ABS**(-salary) AS positive_salary

FROM Employees;

Output:

emp_name	positive_salary

Sai Kiran	45678.50
emp_name	**positive_salary**
Anu Priya	38900.75
Rahul Dev	50234.00

3) MOD(salary, 1000)

Returns remainder after dividing salary by 1000.

SELECT emp_name, **MOD**(salary,1000) AS remainder

FROM Employees;

Output:

emp_name	remainder
Sai Kiran	678.50
Anu Priya	900.75
Rahul Dev	234.00

4) CEIL(salary)

Rounds salary upward.

SELECT emp_name, **CEIL**(salary) AS ceil_salary

FROM Employees;

Output:

emp_name	ceil_salary
Sai Kiran	45679
Anu Priya	38901
Rahul Dev	50234

5) FLOOR(salary)

Rounds salary downward.

SELECT emp_name, **FLOOR**(salary) AS floor_salary

FROM Employees;

Output:

emp_name	floor_salary
Sai Kiran	45678
Anu Priya	38900
Rahul Dev	50234

STRING FUNCTIONS

1) UPPER(emp_name)

Converts text to uppercase.

SELECT **UPPER**(emp_name) AS name_upper

FROM Employees;

Output:

name_upper
SAI KIRAN
ANU PRIYA
RAHUL DEV

2) LOWER(emp_name)

Converts text to lowercase.

SELECT **LOWER**(emp_name) AS name_lower

FROM Employees;

Output:

name_lower
sai kiran

anu priya
name_lower
rahul dev

3) LENGTH(emp_name)

Returns number of characters.

SELECT emp_name, **LENGTH**(emp_name) AS name_length

FROM Employees;

Output:

emp_name	**name_length**
Sai Kiran	9
Anu Priya	9
Rahul Dev	9

4) CONCAT(emp_name, ' - Staff')

Joins strings together.

SELECT **CONCAT**(emp_name, ' - Staff') AS full_label

FROM Employees;

Output:

full_label
Sai Kiran - Staff
Anu Priya - Staff
Rahul Dev - Staff

5) SUBSTRING(emp_name,1,3)

Extracts part of a string.

SELECT emp_name, **SUBSTRING**(emp_name,1,3) AS short_name

FROM Employees;

Output:

emp_name	short_name
Sai Kiran	Sai
Anu Priya	Anu
Rahul Dev	Rah

6) TRIM(' Sai Kiran ')

Removes extra spaces.

SELECT **TRIM**(' Sai Kiran ') AS trimmed_text;

Output:

trimmed_text
Sai Kiran

7) CAST(salary AS CHAR)

Converts number to text.

CAST() is a **type conversion function** used to convert one data type into another.

SELECT emp_name, **CAST**(salary AS CHAR) AS salary_text

FROM Employees;

Output:

emp_name	salary_text
Sai Kiran	45678.50
Anu Priya	38900.75
Rahul Dev	50234.00

Creating Tables with Relationship

- The CREATE TABLE statement is used in SQL to define a new relation by specifying its attributes, data types, and integrity constraints.
- Relationships between tables are established using **foreign key constraints**, which reference the primary key of another relation.
- This mechanism ensures **referential integrity**, meaning that a value in the foreign key attribute must match a value in the referenced primary key.
- Thus, relational databases maintain consistency between related tables.

Example Database

Consider two relations:

- DEPARTMENT
- EMPLOYEE

Each employee works in one department. This represents a **one-to-many relationship**.

Step 1: Creating the DEPARTMENT Table

```
CREATE TABLE department (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(20) NOT NULL,
   location VARCHAR(20)
);
```

- dept_id → Primary Key (uniquely identifies each department)
- NOT NULL → ensures dept_name cannot be empty
- This table is the **parent table**

Step 2: Creating the EMPLOYEE Table with Relationship

```
CREATE TABLE employee (
   emp_id INT PRIMARY KEY,
   name VARCHAR(20) NOT NULL,
   salary DECIMAL(10,2) CHECK (salary > 0),
   dept_id INT,

  FOREIGN KEY (dept_id)
      REFERENCES department(dept_id)
      ON DELETE SET NULL
);
```

- dept_id in EMPLOYEE is a **foreign key**
- It references dept_id in DEPARTMENT
- This establishes a relationship between the two tables
- ON DELETE SET NULL ensures that if a department is deleted, employee's dept_id becomes NULL instead of violating integrity

Sample Data

DEPARTMENT

dept_id	dept_name	location
10	CSE	Block A
20	ECE	Block B

EMPLOYEE

emp_id	name	salary	dept_id
101	Ravi	50000	10
102	Anil	60000	20

Referential Integrity Rule

If we try:

INSERT INTO employee VALUES (103, 'Sai', 45000, 50);

It will fail because:

- dept_id 50 does not exist in department table.
- This protects database consistency.

TYPES OF INTEGRITY CONSTRAINTS

Integrity constraints are rules that maintain **accuracy, consistency, and reliability** of data in a relational database.

1) Entity Integrity

Entity integrity ensures that each row in a table is uniquely identifiable.

Entity Integrity constraints are of two types:

- UNIQUE Constraint
- PRIMARY KEY Constraint

a) UNIQUE Constraint

- A column declared with UNIQUE does not accept duplicate values.
- One table can have multiple UNIQUE constraints.
- UNIQUE column allows NULL values unless declared with NOT NULL.
- Oracle automatically creates a UNIQUE index.

Column Level Example:

```
CREATE TABLE dept
(
 deptno NUMBER(4),
 dname VARCHAR2(20) CONSTRAINT uq_dname_dept UNIQUE,
 loc VARCHAR2(20)
);
INSERT INTO dept VALUES (10,'ACCOUNTING','HYDERABAD');
1 row created.
INSERT INTO dept VALUES (20,'ACCOUNTING','MUMBAI');
ERROR ORA-00001: unique constraint violated.
```

b) **PRIMARY KEY Constraint**

PRIMARY KEY is a candidate key that uniquely identifies each record.

Characteristics:

- Only one primary key per table
- Does not allow NULL values
- Does not allow duplicate values
- Oracle creates a UNIQUE index automatically

- Can be composite (multiple columns)

Column Level Example

CREATE TABLE dept

(

deptno NUMBER(4) CONSTRAINT pk_dept **PRIMARY KEY,**

dname VARCHAR2(20),

loc VARCHAR2(20)

);

ORDER_DETAILS Table:

OrderId	ProdId	Quantity
1000	10	100
1000	11	50
1001	10	20
1001	11	50

OrderId alone → not unique
ProdId alone → not unique
Combination (OrderId, ProdId) → unique

CREATE TABLE order_details

(

ordid NUMBER(4),

prodid NUMBER(4),

qty NUMBER(2),

CONSTRAINT pk_ordid_prodid **PRIMARY KEY**(ordid,prodid)

);

2) Referential Integrity

A referential integrity constraint states that the foreign key value must match the primary key or unique key of another (or same) table.

- Parent / Master Table → Holds Primary Key
- Child / Detail Table → Holds Foreign Key

a) FOREIGN KEY Constraint

- Establishes relationship between tables
- Allows NULL values (unless NOT NULL specified)
- Allows duplicate values
- Default relationship is 1 : M

Column Level Example

Parent Table

```
CREATE TABLE dept
(
 deptno NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
 dname VARCHAR2(20),
 loc VARCHAR2(20)
);
```

Insert Parent Records

Deptno	Dname	Loc
10	Accounting	Hyderabad
20	Research	Mumbai

Child Table

```
CREATE TABLE emp
(
 empno NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
 ename VARCHAR2(20) NOT NULL,
 sal NUMBER(7,2) CONSTRAINT ck_sal_emp CHECK(sal>3000),
 deptno NUMBER(2) CONSTRAINT fk_deptno_emp REFERENCES dept(deptno)
);
```

Insert Records

Empno	Ename	Salary	Deptno	Result
1	Smith	5000	10	Inserted (FK matches PK)
2	Allen	4000	NULL	Inserted (FK allows NULL)
3	Blake	6000	90	Error (FK not matching PK)
4	King	7000	10	Inserted (Duplicates allowed)

3) Self Referential Integrity

If a foreign key refers to the primary key of the **same table**, it is called self referential integrity.

Example: Employee table where Manager is also Employee.

4) Domain Constraints

A domain means the set of valid values assigned to a column.

Domain constraints are handled by:

- Defining proper data type
- NOT NULL constraint
- CHECK constraint

a) NOT NULL Constraint

- Ensures column cannot be left empty
- Mandatory column
- Declared only at column level

Example

```
CREATE TABLE emp
(
 empno NUMBER(4),
 ename VARCHAR2(20) NOT NULL,
 job VARCHAR2(20)
);
```

b) CHECK Constraint

- Validates data based on a condition

- Allows NULL values
- Cannot use ROWNUM, SYSDATE etc.
- Cannot reference another table

Column Level Example

```
CREATE TABLE accounts_master
(
 accno NUMBER(4) PRIMARY KEY,
 acname VARCHAR2(20) NOT NULL,
 balance NUMBER(11,2) CONSTRAINT ck_bal_accts CHECK(balance>1000)
);
INSERT INTO accounts_master VALUES (1,'A',500);
```

ERROR: check constraint violated

Table Level Example

Rule: End_date > Start_date

```
CREATE TABLE managers
(
 mgrno NUMBER(4) PRIMARY KEY,
 mname VARCHAR2(20) NOT NULL,
 start_date DATE,
 end_date DATE,
 CONSTRAINT ck_mgr CHECK(end_date > start_date)
);
```

5) DEFAULT Option

If a column is declared with DEFAULT, Oracle inserts default value when not provided.

```
CREATE TABLE emp
(
 empno NUMBER(4),
 ename VARCHAR2(20),
 hiredate DATE DEFAULT SYSDATE
);
```

6) Adding Constraints (ALTER TABLE)

ALTER TABLE emp55

ADD CONSTRAINT pk_emp55 PRIMARY KEY(empno);

7) Dropping Constraints

ALTER TABLE emp55 **DROP** CONSTRAINT pk_emp55;

Important Notes:

- Primary key cannot be dropped if referenced by foreign key.
- CASCADE option removes referencing foreign keys.
- Parent table cannot be dropped if referenced.

JOINS

- In OLTP databases, tables are normalized and data is organized into more than one table such as CUSTOMER, PRODUCT, SUPPLIER, EMP, DEPT, etc.
- JOIN is an operation that combines rows from two or more tables or views.
- Oracle performs JOIN operation when more than one table is listed in the FROM clause.
- Tables participating in JOIN operation must share a meaningful relationship.
- To join N tables, minimum N-1 join conditions are required.

TYPES OF JOINS

1. INNER JOIN
2. NON-EQUI JOIN
3. SELF JOIN
4. OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
5. CROSS JOIN
6. NATURAL JOIN

SAMPLE TABLES CREATION

```
CREATE TABLE dept (
 deptno NUMBER PRIMARY KEY,
 dname  VARCHAR2(20),
 loc    VARCHAR2(20)
);

CREATE TABLE emp (
 empno NUMBER PRIMARY KEY,
 ename VARCHAR2(20),
 sal   NUMBER,
 deptno NUMBER,
 mgr NUMBER
);
```

```
CREATE TABLE salgrade (
 grade NUMBER,
 losal NUMBER,
 hisal NUMBER
);
```

INSERT SAMPLE DATA

```
INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO dept VALUES (20,'SALES','CHICAGO');
INSERT INTO dept VALUES (30,'HR','BOSTON');

INSERT INTO emp VALUES (1,'A',3000,10,NULL);
INSERT INTO emp VALUES (2,'B',1500,20,1);
INSERT INTO emp VALUES (3,'C',2500,10,1);
INSERT INTO emp VALUES (4,'D',1000,NULL,NULL);

INSERT INTO salgrade VALUES (1,0,1500);
INSERT INTO salgrade VALUES (2,1501,3000);
```

1) INNER JOIN (EQUI JOIN)

• INNER JOIN is performed based on common columns.
• To perform INNER JOIN, there should be a common column in joining tables, but the column names need not be the same.
• Parent-child relationship between tables is not mandatory.
• INNER JOIN uses = operator, so it is also called EQUI JOIN.
• It returns only matching rows from both tables that satisfy the join condition.
• To join N tables, minimum N-1 join conditions are required.

Query:

```
SELECT e.empno, e.ename, d.dname, d.loc FROM emp e, dept d WHERE e.deptno = d.deptno;
```

Output:

EMPNO	ENAME	DNAME	LOC
1	A	ACCOUNTING	NEW YORK

EMPNO	ENAME	DNAME	LOC
2	B	SALES	CHICAGO
3	C	ACCOUNTING	NEW YORK

(Employee 4 not shown because no department)

2) **NON-EQUI JOIN**

• When the join condition is based on equality operator (=), it is called Equi Join.
• When the join condition is based on operators other than =, it is called Non-Equi Join.
• It generally uses BETWEEN, >, < operators.
• Used when values fall within a specified range.
• Example: Matching employee salary with salary grade range.

Query:

SELECT e.empno, e.ename, e.sal, s.gradeFROM emp e, salgrade s WHERE e.sal BETWEEN s.losal AND s.hisal;

Output:

EMPNO	ENAME	SAL	GRADE
1	A	3000	2
2	B	1500	1
3	C	2500	2
4	D	1000	1

3) SELF JOIN

• Joining a table to itself is called SELF JOIN.
• Same table must be used with different aliases.
• Used when table has self reference (manager).
• It is an equi join within same table.

Query:

SELECT e.empno, e.ename, m.ename AS manager FROM emp e, emp m **WHERE** e.mgr = m.empno;

Output:

EMPNO	ENAME	MANAGER
2	B	A
3	C	A

4) LEFT OUTER JOIN

• Returns all rows from left table.
• Matching rows from right table.
• Unmatched rows show NULL.
• (+) operator used on right side (old Oracle syntax).

Query:

SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno(+);

Output:

EMPNO	ENAME	DNAME
1	A	ACCOUNTING
2	B	SALES
3	C	ACCOUNTING
4	D	NULL

5) RIGHT OUTER JOIN

• Returns all rows from right table.
• Matching rows from left table.
• Unmatched rows show NULL.
• (+) used on left side.

Query:

SELECT e.empno, e.ename, d.dname FROM emp e, dept d WHERE e.deptno**(+)** = d.deptno;

Output:

EMPNO	ENAME	DNAME
1	A	ACCOUNTING

EMPNO	ENAME	DNAME
2	B	SALES
3	C	ACCOUNTING
NULL	NULL	HR

6) FULL OUTER JOIN

• Returns all rows from both tables.
• Includes matched and unmatched rows.
• Unmatched rows contain NULL.
• Supported from Oracle 9i onwards.

Query:

SELECT e.empno, e.ename, d.dnameFROM emp e FULL OUTER JOIN dept d ON e.deptno = d.deptno;

Output:

EMPNO	ENAME	DNAME
1	A	ACCOUNTING
2	B	SALES
3	C	ACCOUNTING
4	D	NULL
NULL	NULL	HR

7) CROSS JOIN

• Returns Cartesian product of two tables.
• Each row of first table joins with every row of second table.
• If 4 rows and 3 rows → 12 rows result.
• Happens when no join condition.

Query:

SELECT e.ename, d.dname FROM emp e, dept d;

Output:

(4 employees × 3 departments = 12 rows)

Example rows:

ENAME	DNAME
A	ACCOUNTING
A	SALES
A	HR
B	ACCOUNTING
...	...

8) NATURAL JOIN

- Works only if common column name same.
- No need to specify join condition.
- Automatically joins based on same column name.
- Similar to Equi Join.

Query:

SELECT empno, ename, dname FROM emp **NATURAL JOIN** dept;

(Output same as INNER JOIN)

SET OPERATORS

1) **UNION**

• Combines results of two SELECT statements.
• Removes duplicates.
• Sorts result automatically.

Example:

SELECT deptno FROM emp **UNION SELECT** deptno FROM dept;

2) **UNION ALL**

• Same as UNION.
• Includes duplicates.
• Does not sort automatically.

Example:

SELECT deptno FROM emp **UNION ALL SELECT** deptno FROM dept;

3) INTERSECT

• Returns common values from two queries.
• Removes duplicates.
• Shows only matching rows.

Example:

SELECT deptno FROM emp **INTERSECT SELECT** deptno FROM dept;

4) MINUS

• Returns rows present in first query but not in second.
• Removes duplicates.
• Order by must be in last query.

Example:

SELECT deptno FROM dept **MINUS SELECT** deptno FROM emp;

SUB QUERIES

- A Subquery is a query written inside another SQL query.
- It is also called Inner Query or Nested Query.
- The outer query is called Main Query or Parent Query.
- The result of inner query is passed to the outer query for further processing.

• Inner query must always be SELECT statement.
• Outer query can be SELECT, INSERT, UPDATE or DELETE.
• Subqueries are mostly used in WHERE, HAVING and FROM clauses.
• Subqueries improve readability and simplify complex queries.

SAMPLE TABLE

```
CREATE TABLE emp (
  empno NUMBER PRIMARY KEY,
  ename VARCHAR2(20),
  job  VARCHAR2(20),
  sal  NUMBER,
  deptno NUMBER,
  mgr NUMBER
);
INSERT INTO emp VALUES (7369,'SMITH','CLERK',800,20,7902);
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',1600,30,7698);
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',1250,30,7698);
INSERT INTO emp VALUES (7566,'JONES','MANAGER',2975,20,7839);
INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',2850,30,7839);
INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',3000,20,7566);
INSERT INTO emp VALUES (7839,'KING','PRESIDENT',5000,10,NULL);
INSERT INTO emp VALUES (7902,'FORD','ANALYST',3000,20,7566);
```

TYPES OF SUBQUERIES

1. Single Row Subquery
2. Multi Row Subquery
3. Nested Subquery
4. Multi Column Subquery
5. Co-related Subquery

1) SINGLE ROW SUBQUERY

- If the inner query returns only one row, it is called Single Row Subquery.
- It uses single row operators like =, >, <, >=, <=, <>.
- It is used when comparison is done with a single value.
- Aggregate functions like MAX, MIN, AVG usually return single value.

Example:

Display employee earning maximum salary.

SELECT ename

FROM emp

WHERE sal = (**SELECT** MAX(sal) FROM emp);

Output:

ENAME
KING

2) MULTI ROW SUBQUERY

- If inner query returns more than one row, it is called Multi Row Subquery.
- It uses operators like IN, NOT IN, ANY, ALL.
- Single row operators cannot be used here.
- Used when comparison is done with multiple values.

Example:

Display employees whose job is same as SMITH or BLAKE.

SELECT ename, job

FROM emp

WHERE job **IN** (SELECT job FROM emp

WHERE ename **IN** ('SMITH','BLAKE'));

Output:

ENAME	JOB

SMITH	CLERK
ENAME	**JOB**
JONES	MANAGER
BLAKE	MANAGER

3) ANY OPERATOR

- ANY compares a value with each value returned by subquery.
- It returns TRUE if condition matches at least one value.
- It must be used with comparison operators.
- Useful when partial matching is required.

Example:

Employees earning more than ANY salesman salary.

SELECT ename, sal

FROM emp

WHERE sal > **ANY** (SELECT sal FROM emp WHERE job='SALESMAN');

Output:

ENAME	**SAL**
ALLEN	
JONES	
BLAKE	
SCOTT	
KING	
FORD	

4) ALL OPERATOR

- ALL compares a value with every value returned by subquery.
- It returns TRUE only if condition satisfies all values.
- It is stricter than ANY operator.
- Used for complete comparison.

Example:

Employees earning more than ALL salesman salary.

SELECT ename, sal

FROM emp

WHERE sal > **ALL** (SELECT sal FROM emp WHERE job='SALESMAN');

Output:

ENAME
JONES
BLAKE
SCOTT
KING
FORD

5) NESTED SUBQUERY

- When a subquery contains another subquery, it is called Nested Subquery.
- It can be nested up to multiple levels.
- Used to solve complex queries like second highest salary.
- Execution happens from inner-most query.

Example:

Employee earning second highest salary.

SELECT ename

FROM emp

WHERE sal = (SELECT MAX(sal)

FROM emp

WHERE sal < (SELECT MAX(sal) FROM emp));

Output:

ENAME
SCOTT

FORD

Prepared by: V.Ragubathy

6) MULTI COLUMN SUBQUERY

- If subquery returns more than one column, it is called Multi Column Subquery.
- Outer query must match number of columns.
- Used for composite comparisons.
- Often used with GROUP BY.

Example:

Employees earning maximum salary in their department.

```
SELECT ename, deptno, sal
FROM emp
WHERE (deptno, sal) IN
    (SELECT deptno, MAX(sal)
    FROM emp
     GROUP BY deptno);
```

Output:

ENAME	DEPTNO	SAL
KING	10	5000
SCOTT	20	3000
FORD	20	3000
BLAKE	30	2850

7) CO-RELATED SUBQUERY

- If subquery refers to column of outer query, it is called Co-related Subquery.
- It executes once for each row of outer query.
- Inner query depends on outer query value.
- Used for row-by-row comparison.

Example:

Employees earning more than average salary of their department.

```
SELECT ename, sal, deptno
FROM emp e
```

WHERE sal > (SELECT AVG(sal)

FROM emp

WHERE deptno = e.deptno);

Output:

ENAME	SAL	DEPTNO
JONES		
SCOTT		
FORD		
KING		
BLAKE		

8) EXISTS Operator

- EXISTS checks whether subquery returns any rows.
- If at least one row exists, condition becomes TRUE.
- It returns only TRUE or FALSE.
- It is faster than IN in many cases.

Example:

Display departments that have employees.

SELECT DISTINCT deptno

FROM emp e

WHERE EXISTS (SELECT *

FROM emp

WHERE deptno = e.deptno);

Output:

DEPTNO
10
20
30

Views in SQL

- **Views in SQL are considered as a virtual table. A view also contains rows and columns.**
- **To create the view, we can select the fields from one or more tables present in the database.**
- **A view can either have specific rows based on certain condition or all the rows of a table.**

Sample table:Student_Detail

(Updatable and Non-Updatable)

- A View is a virtual table based on a SELECT query.
- It does not store data physically.
- It retrieves data from one or more base tables.
- Views are mainly used for security and data abstraction.

SAMPLE TABLE

```
CREATE TABLE emp (
  empno NUMBER PRIMARY KEY,
  ename VARCHAR2(20),
  sal NUMBER,
  deptno NUMBER
);

INSERT INTO emp VALUES (101,'RAJU',4000,10);
INSERT INTO emp VALUES (102,'RAVI',3000,20);
INSERT INTO emp VALUES (103,'ANU',5000,10);
INSERT INTO emp VALUES (104,'SITA',3500,30);
```

UPDATABLE VIEW

- A view on which INSERT, UPDATE and DELETE operations can be performed is called Updatable View.
- Changes made through the view reflect in the base table.
- Generally, simple views based on a single table are updatable.
- The view must not contain GROUP BY, DISTINCT, aggregate functions or joins.

Create Updatable View

```
CREATE VIEW emp_v AS
SELECT empno, ename, sal, deptno
FROM emp;
```

Select from View

```
SELECT * FROM emp_v;
```

Output

EMPNO	ENAME	SAL	DEPTNO
101	RAJU	4000	10
102	RAVI	3000	20
103	ANU	5000	10

EMPNO	ENAME	SAL	DEPTNO
104	SITA	3500	30

Insert Through View

INSERT INTO emp_v VALUES (105,'KIRAN',4500,20);

Update Through View

UPDATE emp_v
SET sal=4800
WHERE empno=105;

Delete Through View

DELETE FROM emp_v
WHERE empno=105;

These changes affect the base table EMP.

NON-UPDATABLE VIEW

- A view on which DML operations cannot be performed is called Non-Updatable View.
- Views containing JOIN, GROUP BY, DISTINCT, aggregate functions are not updatable.
- Such views are mainly used for reporting purposes.
- They provide summarized or combined data.

Example (Aggregate View)

CREATE VIEW dept_sum_v AS
SELECT deptno, SUM(sal) AS total_sal
FROM emp
GROUP BY deptno;

Query

SELECT * FROM dept_sum_v;

Output

DEPTNO	TOTAL_SAL
10	9000
20	3000
30	3500

X- Cannot perform INSERT, UPDATE, DELETE on this view.

TEXT BOOKS:

1. Database Management Systems, 3rd edition, Raghurama Krishnan, Johannes Gehrke, TMH

2. Database System Concepts,6th edition, Silberschatz, Korth, Sudarsan,TMH

REFERENCE BOOKS:

1. Introduction to Database Systems, 8thedition, C J Date, Pearson.

2. Database Management System, 6th edition, RamezElmasri, Shamkant B. Navathe, Pearson

REFERENCE WEBSITE:

1. https://www.w3schools.in/sql/database-concepts

2. https://www.javatpoint.com/dbms-tutorial

3. https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system- set-1/

4. https://nptel.ac.in/courses/106/105/106105175/

UNIT-IV

.

1. Schema Refinement:

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is **decomposition.**

Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

Anomalies or problems facing without normalization(problems due to redundancy) :

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

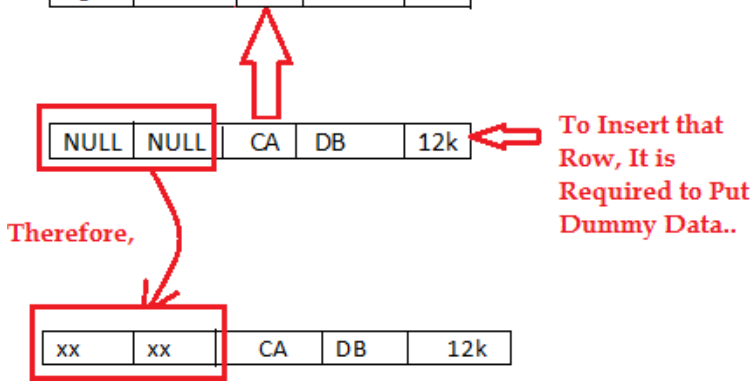
1. **insertion anomalies :** It may not be possible to store some information unless some other information is stored as well.
2. **redundant storage:** some information is stored repeatedly
3. **update anomalies:** If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.
4. **deletion anomalies:** It may not be possible to delete some information without losing some other information as well.

Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.

Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

InsertionAnomalies: New course is introduced C4, But no student is there who is having C4 subject.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k



Because of insertion of some data, It is forced to insert some other dummy data.

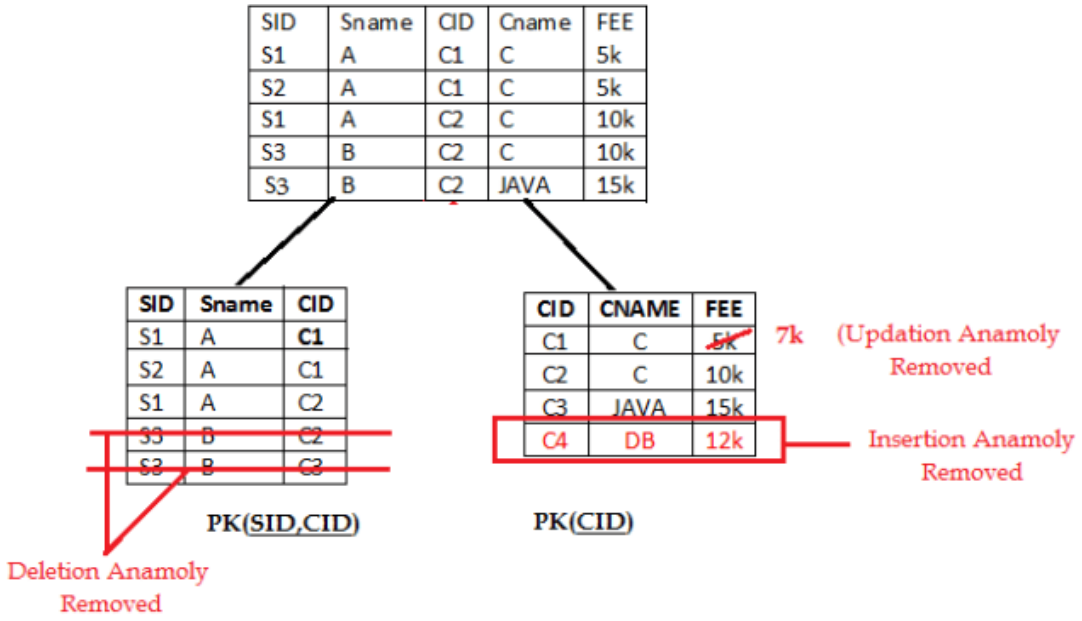
Deletion Anomaly :

Deletion of S3 student cause the deletion of course.

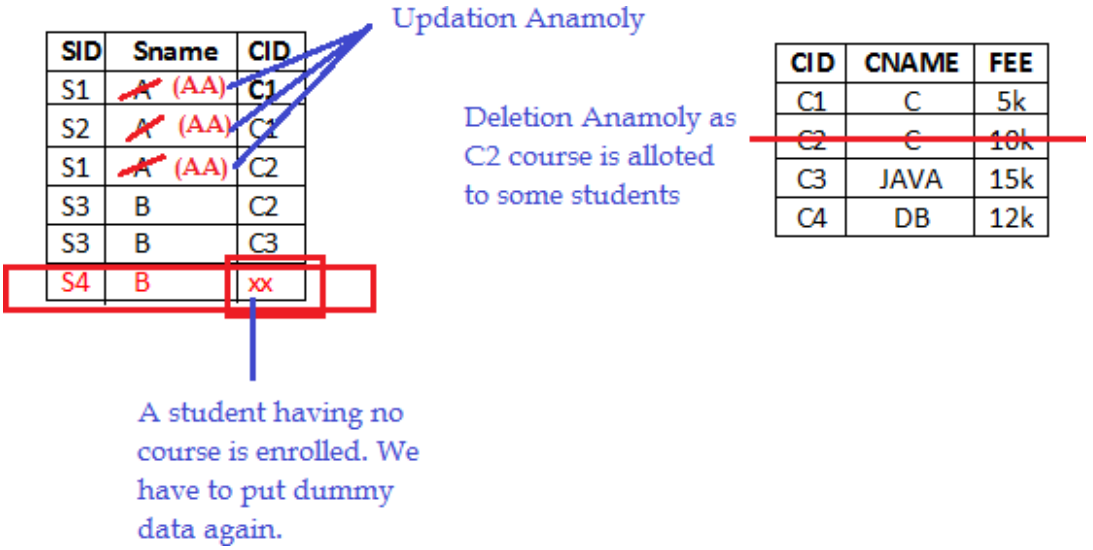
Because of deletion of some data forced to delete some other useful data.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

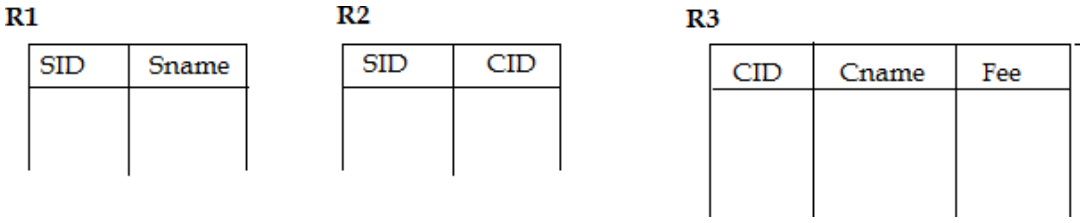
Solutions To Anomalies : Decomposition of Tables – Schema Refinement



There are some Anomalies in this again –



What is the Solution ??
Solution : decomposing into relations as shown below



TO AVOID REDUNDANCY and problems due to redundancy, we use refinement technique called **DECOMPOSITION.**

Decomposition:- Process of decomposing a larger relation into smaller relations.

Each of smaller relations contain subset of attributes of original relation.

Functional dependencies:

Functional dependency is a relationship that exist when one attribute uniquely determines another attribute.

Functional dependency is a form of integrity constraint that can identify schema with redundant storage problems and to suggest refinement.

A functional dependency A B in a relation holds true if two tuples having the same value of attribute A also have the same value of attribute B

IF t1.X=t2.X then t1.Y=t2.Y where t1,t2 are tuples and X,Y are attributes.

Reasoning about functional dependencies:

Armstrong Axioms :

Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:

Primary axioms:

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

secondary or derived axioms:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Attribute closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

NOTE:

To find attribute closure of an attribute set-

1) add elements of attribute set to the result set.

2) recursively add elements to the result set which can be functionally determined from the elements of result set.

Types of functional dependencies:

1) **Trivial functional dependency:-**If X □ Y is a functional dependency where Y subset X, these type of FD's called as trivial functional dependency.

2) Non-trivial functional dependency:-If X □ Y and Y is not subset of X then it is called non-trivial functional dependency.

3) Completely non-trivial functional dependency:-If X □ Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency.

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

Candidate Key:
Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
Consider student table: student(sno, sname,sphone,age)
we can take **sno** as candidate key. we can have more than 1 candidate key in a table. types of candidate keys:
1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

Super Key:
Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(sno, sname,sphone,age) we can take sno, (sno, sname) as super key

Finding candidate keys problems:

Example 1: Find candidate keys for the relation R(ABCD) having following FD's

AB→CD, C→A, D→A

Solution:

AB^+ = {ABCD} A and B are prime attributes

C→A replace A by c

BC^+ = {ABCD} A and C are prime attributes ($A^+ = A^+ = \{AC\}$)

D→B replace B by D

AD^+ = {ABCD} A and D are prime attributes ($D^+ = \{BD\}$)

CD^+ = {ABCD} (replacing A by C in AD)

AB, BC, CD, AD are candidate keys.

Example 2: Find candidate keys for R(ABCDE) having following FD's

A→BC,CD→E,B→D, E→A

Solution:

A^+ = {ABCDE} A is candidate key and prime attribute

E→A so replace A by E

E^+ = {ABCDE} E is candidate key and prime attribute

CD→E replace E by CD

CD^+ = {ABCDE} ($C^+ = C$ and $D^+ = D$) no proper subset of CD is superkey. so CD is candidate key

B→D

BC^+ = {ABCDE} ($B^+ = BD$) BC is candidate key

A, E, CD, BC are candidate keys

Normalization:

Normalization is a process of designing a consistent database with minimum redundancy which support data integrity by grating or decomposing given relation into smaller relations preserving constraints on the relation.

Normalisation removes data redundancy and it will helps in designing a good data base which involves a set of normal forms as follows -

1) First normal form(1NF)
2) Second normal form(2NF)
3) Third normal form(3NF)
4) Boyce coded normal form(BCNF)
5) Forth normal form(4NF)
6) Fifth normal form(5NF)
7) Sixth normal form(6NF)
8) Domain key normal form.

Normal Forms

1st Normal Form	No repeating data groups
2nd Normal Form	No partial key dependency
3rd Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4th Normal Form	No multi-valued dependency
5th Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

1) First normal form: A relation is said to be in first normal form if it contains all atomic values or single values.

Example:

Domain	**Courses**
Programming	C , java
Web designing	HTML , PHP

The above table consist of multiple values in single columns which can be reduced into atomic values by using first normal form as follows-

Domain	**Courses**
Programming	C
Programming	Java
Web designing	HTML
Web designing	PHP

2) Second normal form: A relation is said to be in second normal form if it is in first normal form without any partial dependencies.

In second normal form non-prime attributes should not depend on proper subset of key attributes.

Example:

Student id	**Student name**	**Project Id**	**Project name**

Here (student id, project id) are key attributes and (student name, project name) are non-prime attributes. It is decomposed as-

Student id	Student name	Project id

Project id	Project name

3) Third normal form: A relation is said to be in third normal form , if it is already in second normal form and no transitive dependencies exists.

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Student id	Student name	City	country	ZIP

It is decomposed as:

Student id	Student name	ZIP

ZIP	city	country

4) **Boyce normal form:** It is an extension of third normal form where in a functional dependency X┌ A , X must be a super key.

A relation is in BCNF if in every non-trivial functional dependency X –> Y, X is a super key.

5) **fourth normal form:** A relation is said to be in fourth normal form if it is in third normal form and no multi value dependencies should exist between attributes.

Note: In some cases multi value dependencies may exist not more than one time in a given relation.

6) **fifth normal form:** fifth normal form is related to join dependencies.

A relation R is said to be in fifth normal form if for every join dependency JD join {R_1 , R_2 ,…….R_N } that holds over relation R one of the following statements must be true-

1)R_i =R for some i

2) the join dependency is implied by the set of those functional dependency over relation R in which the left side is key attribute for R.

NOTE: if the relation schema is a third normal form and each of its keys consist of single attribute, we can say that it can also be in fifth normal form.

A join dependency JD join {R_1, R_2, ……R_N} is said to hold for a relation R if R_1,R_2…..R_N this decomposition is a loss less join decomposition of R.

When a relation is in forth normal form and decompose further to eliminate redundancy and anomalies due to insert or update or delete operation, there should not be any loss of data or should not create a new record when the decompose tables are rejoin.

7) **Domain key normal form:** A domain key normal form keeps a constraint that every constraint on the relation is a logical sequence of definition of keys and domains.

8) **Sixth normal form:** A relation is said to be in sixth normal form such that the relation R should not contain any non-trivial join dependencies.

Also sixth normal form considers temporal dimensions(time) to the relational model.

Key Points related to normal forms –

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

problems on normal forms:

Problem 1:

Find the highest normal form in R (A, B, C, D, E) under following functional dependencies. ABC --> D
CD --> AE

Solution:

Important Points for solving above type of question.
1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.
Candidate keys in given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2 NF. S**o, the highest normal form is 1 NF.**

problem 2:

Find the highest normal form of a relation R(A,B,C,D,E) with

FD set as

{BC-
>D,
AC-
>BE,

B->E}

Step 1:

As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2:

Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

Step 3:

The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

Decomposition: It is the process of splitting original table into smaller relations such that attribute sets of two relations will be the subset of attribute set of original table.

Rules of decomposition:

If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1) Union of two smaller subsets of attributes gives all attributes of 'R'.

   R1(attributes)UR2(attributes)=R(attributes)

2) Both relations interaction should not give null value.

   R1(attributes)∩R2(attributes)!=null

3) Both relations interaction should give key attribute.

   R1(attribute)∩R2(attribute)=R(key attribute)

Properties of decomposition:

Lossless decomposition: while joining two smaller tables no data should be lost and should satisfy all the rules of decomposition. No additional data should be generated on natural join of decomposed tables.

example 2 for loseless decomposition:

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

→

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B;\ C \rightarrow B$

A	C
1	3
4	6
7	8

⋈

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Lossy join decomposition: if information is lost after joining and if do not satisfy any one of the above rules of decomposition.

example 1:

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

→

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B;\ C \rightarrow B$

A	B
1	2
4	5
7	2

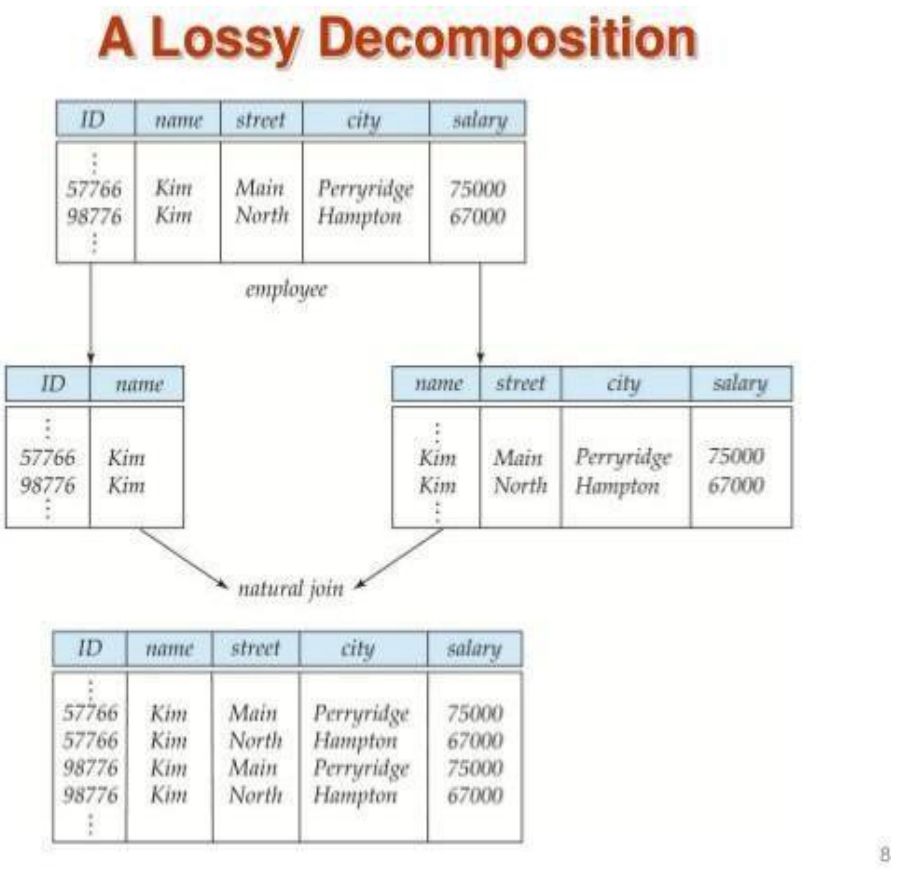
⋈

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

example 2:



In above examples, on joining decomposed tables, extra tuples are generated. so

it is lossy join decomposition.

Dependency preservation: functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables.

Dependency Preservation

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD).....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then there can be three cases:

f1 U f2 = F ----- > Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.

example for dependency preservation:

Dependency preservation

Example:
R=(A, B, C), F={A→B, B→C}

Decomposition of R: R1=(A, B) R2=(B, C)
Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold: F1={A→B, A→A, B→B, AB→AB}
In R2 the following dependencies hold: F2= {B→B, C→C, B→C, BC→BC}

F'= F1' ∪ F2' = {A→B, B→C, trivial dependencies}

In F' all the original dependencies occur, so this decomposition preserves dependencies.

lack of redundancy: It is also known as repetition of information. The proper decomposition should not suffer from any data redundancy.

TEXT BOOKS:

1. Database System Concepts, Silberschatz, Korth, Sudarsan,TMH.

REFERENCE BOOKS:

1. Fundamentals of Database Systems, Elmasri Navathe Pearson Education.

UNIT-V

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durabilit

1. Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A- 100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

2. Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

3. Isolation

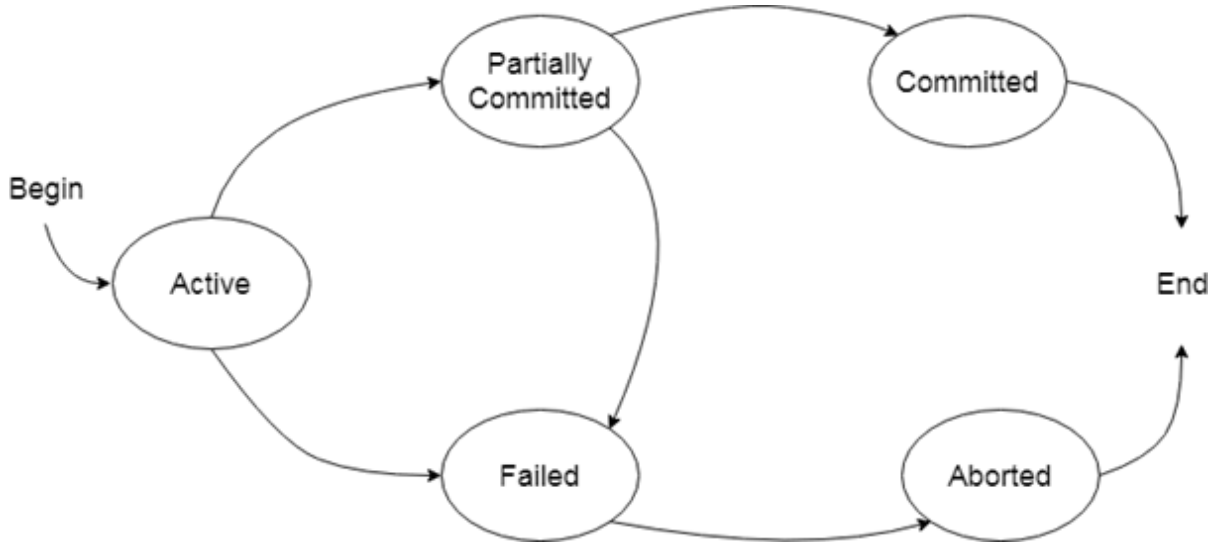
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

4. Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

DBMS Concurrency Control: Timestamp & Lock-Based Protocols

What is Concurrency Control?

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other.

It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

- Lock-Based Protocols
- Two Phase Locking Protocol
- Timestamp-Based Protocols
- Validation-Based

Protocols Lock-based Protocols

Lock Based Protocols in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking ora isolating a particular transaction to a single user.

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Binary Locks: A Binary lock on a data item can either locked or unlocked states.

Shared/exclusive: This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared Lock (S):

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive Lock (X):

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

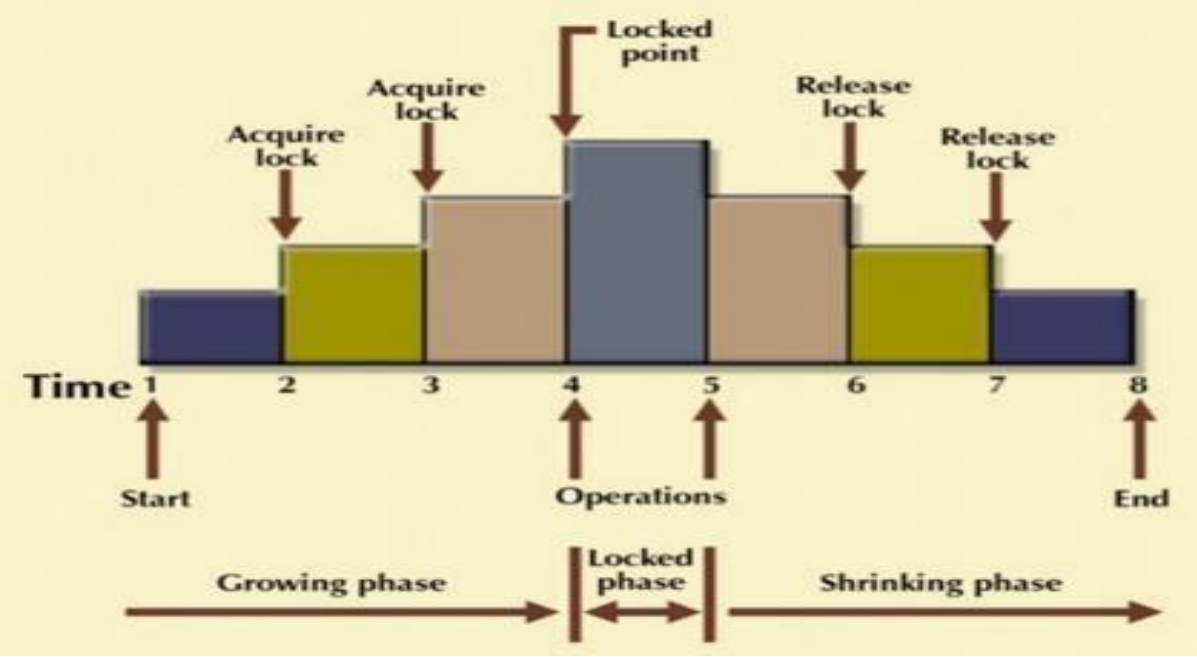
Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously.

Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.

This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Example:

T1	T2
L1(A)	L2(A)
R1(A)	R2(A)
A=A+100	A=2*A
W1(A)	W2(A)
L1(B)	l2(B)
U1(A)	U2(A)
R1(B)	R2(B)
B=B+100	B=2*B
W1(B)	W2(B)
U1(B)	U2(B)

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.

The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Example:

Suppose there are there transactions T1, T2, and T3.

T1 has entered the system at time 0010

T2 has entered the system at 0020

T3 has entered the system at 0030

Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:

Starvation is possible if the same transaction is restarted and continually aborted

Starvation

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:
    - If W_TS(X) >TS(Ti) then the operation is rejected.
    - If W_TS(X) <= TS(Ti) then the operation is executed.
    - Timestamps of all the data items are updated.
2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:
    - If TS(Ti) < R_TS(X) then the operation is rejected.
    - If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

Example

10	20	30	40
T1	T2	T3	T4
R1(A)			
W1(A)			
		R3(A)	
	R2(A)		W4(A)

RTS(A)=0

Update T1=10 0<10(T)

Update T3=30 10<30(T)

Update T2=20 30<20(F)

RTS(A)=30

WTS(A)=0

Update T1=10 0<10(T)

Update T4=40 40<10(F)

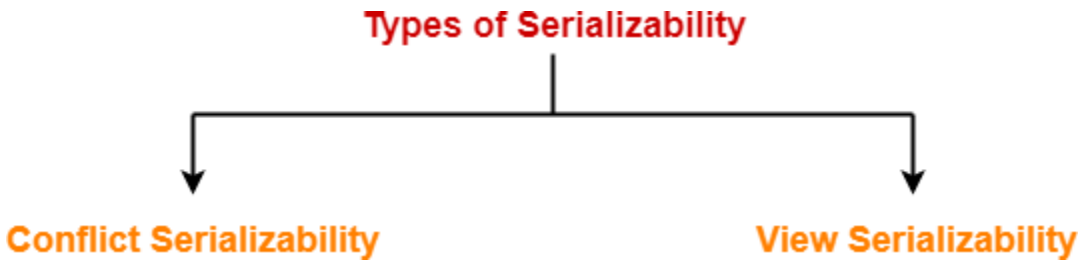
WTS(A)=40

Serializability in DBMS-

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

Types of Serializability-

Serializability is mainly of two types-



Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

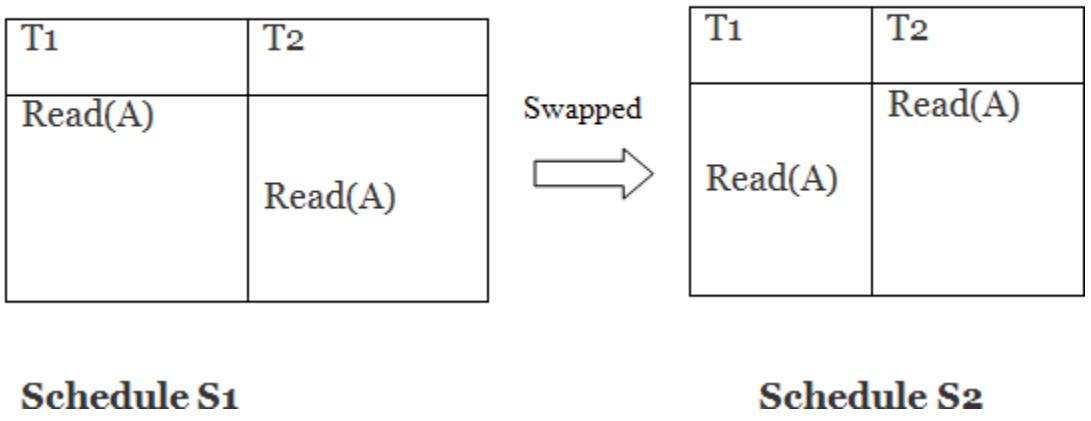
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.

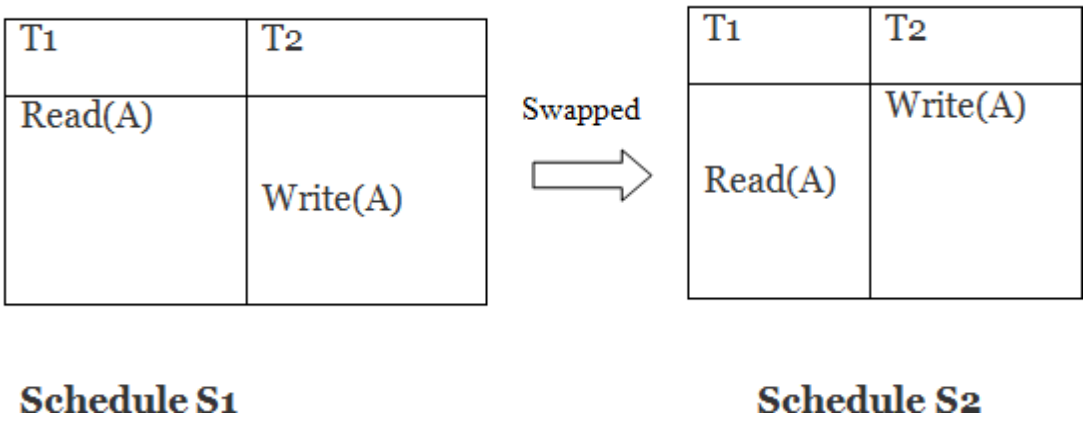
1. T1: Read(A) T2: Read(A)



Schedule S1 **Schedule S2**

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



Schedule S1 **Schedule S2**

Here, S1 ≠ S2. That means it is conflict.

Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:

Non-serial schedule

T1	T2
Read(A) Write(A)	
	Read(A) Write(A)
Read(B) Write(B)	
	Read(B) Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A) Write(A) Read(B) Write(B)	Read(A) Write(A) Read(B) Write(B)

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:

    1. Database
    2. Area
    3. File
    4. Record

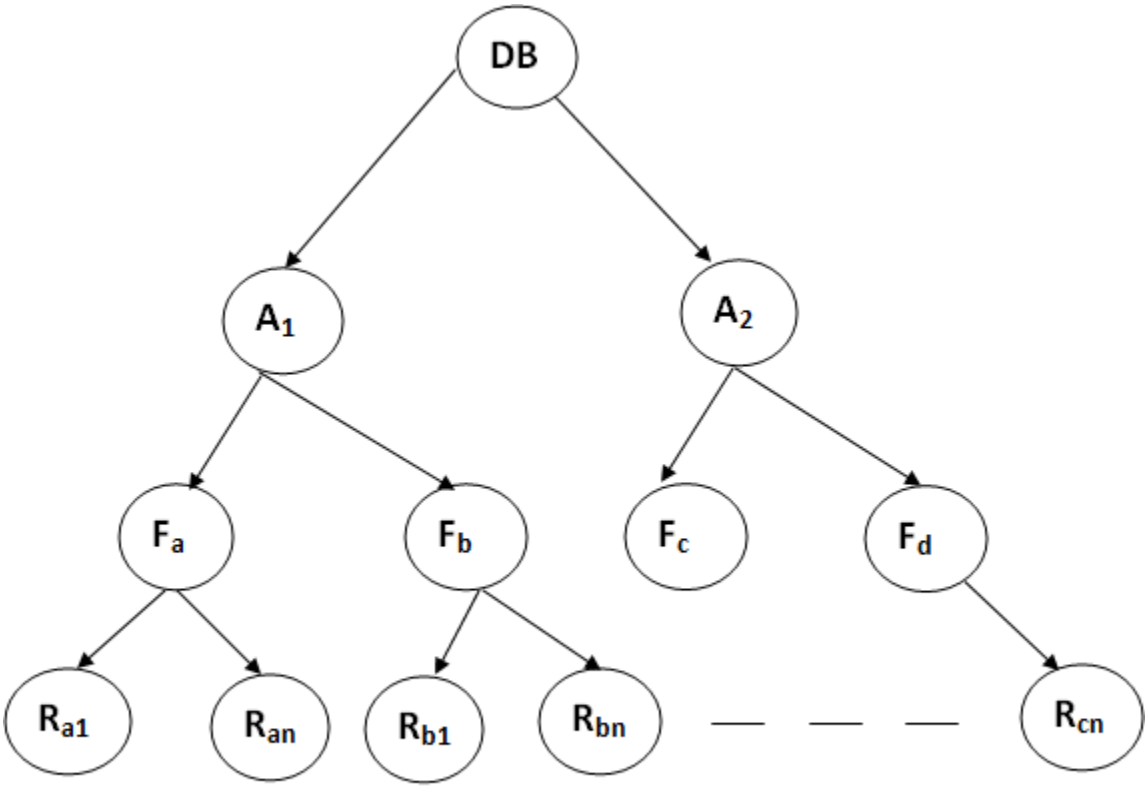


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields.

There are three additional lock modes with multiple granularity:

Intention Mode Lock

Intention-shared (IS): It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

Compatibility Matrix with Intention Lock Modes: The below table describes the compatibility matrix for these lock modes:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

It uses the intention lock modes to ensure serializability. It requires that if a transaction attempts to lock a node, then that node must follow these protocols:

- Transaction T1 should follow the lock-compatibility matrix.
- Transaction T1 firstly locks the root of the tree. It can lock it in any mode.
- If T1 currently has the parent of the node locked in either IX or IS mode, then the transaction T1 will lock a node in S or IS mode only.
- If T1 currently has the parent of the node locked in either IX or SIX modes, then the transaction T1 will lock a node in X, SIX, or IX mode only.
- If T1 has not previously unlocked any node only, then the Transaction T1 can lock a node.
- If T1 currently has none of the children of the node-locked only, then Transaction T1 will unlock a node.

Observe that in multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

- If transaction T1 reads record R_{a9} in file F_a, then transaction T1 needs to lock the database, area A_1 and file F_a in IX mode. Finally, it needs to lock R_{a2} in S mode.
- If transaction T2 modifies record R_{a9} in file F_a, then it can do so after locking the database, area A_1 and file F_a in IX mode. Finally, it needs to lock the R_{a9} in X mode.
- If transaction T3 reads all the records in file F_a, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock F_a in S mode.
- If transaction T4 reads the entire database, then T4 needs to lock the database in S mode.

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have

to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.

- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).

- But the value for account A remains $350 for transaction TY as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, TX and TY, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

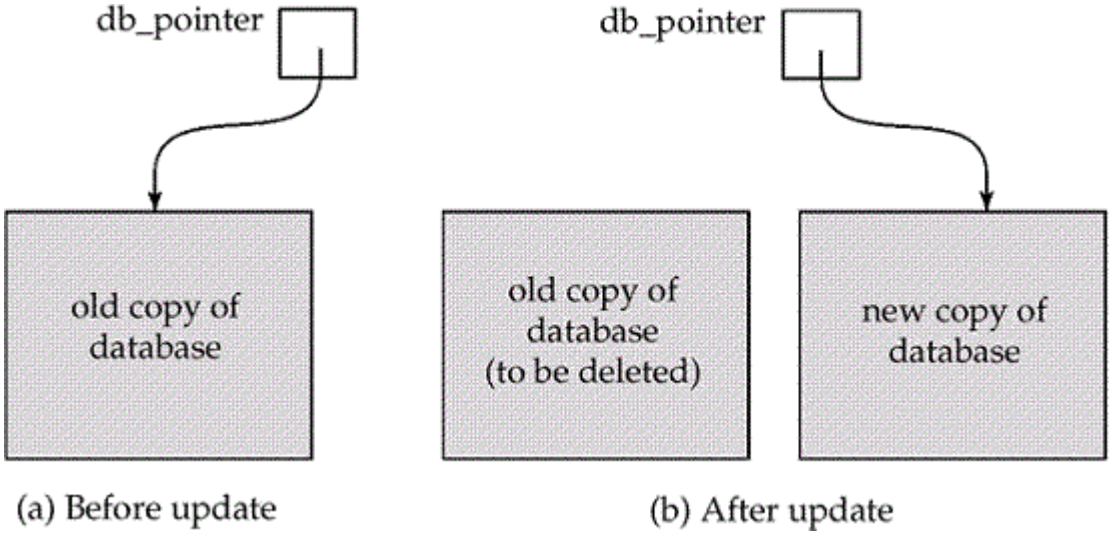
- At time t1, transaction TX reads the value from account A, i.e., $300.
- At time t2, transaction TY reads the value from account A, i.e., $300.
- At time t3, transaction TY updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction TY writes the updated value, i.e., $400.
- After that, at time t5, transaction TX reads the available value of account A, and that will be read as $400.
- It means that within the same transaction TX, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction TY, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Implementation Of Atomicity And Durability

ACID properties are an important concept for databases. The acronym stands for Atomicity, Consistency, Isolation and Durability.

The Shadow-Database Scheme

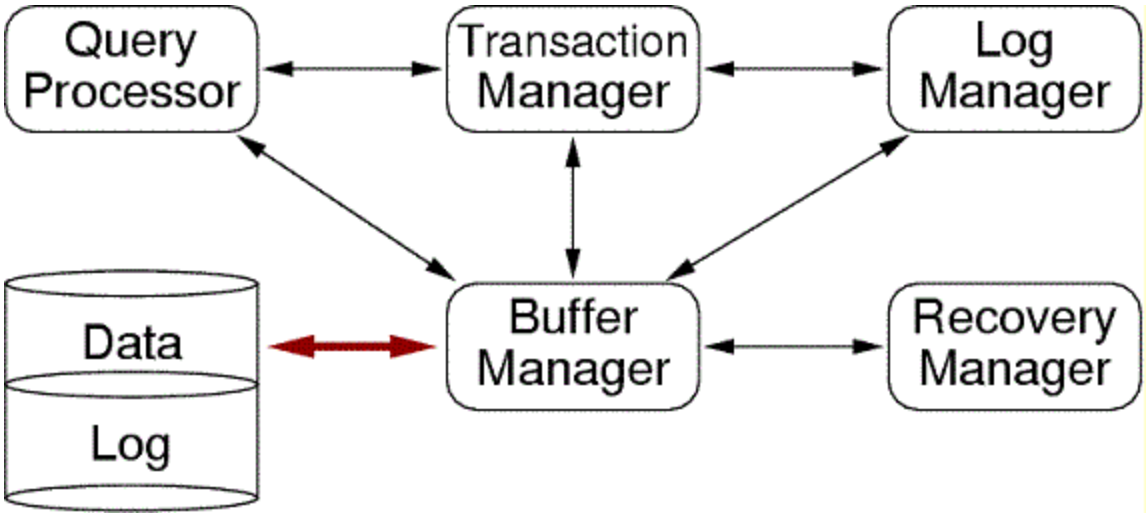


Assumes one transaction at a time.

Useful for text editors, but extremely inefficient for large databases: executing a single transaction requires copying the entire database.

Architecture For Atomicity/Durability

How does a DBMS provide for atomicity/durability?



TEXT BOOKS:

1. Database System Concepts, Silberschatz, Korth, Sudarsan,TMH.

REFERENCE BOOKS:

1. Fundamentals of Database Systems, Elmasri Navathe Pearson Education.