

DBMS

UNIT 1

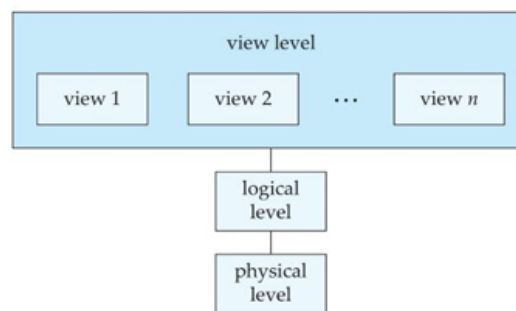
- 1) For the database system to be usable, it must retrieve data efficiently. The need of efficiency has led designers to use complex data structures to represent data in the database. Developers hide this complexity from the database system users through several levels of abstraction. Explain those levels of abstraction in detail with example.

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.

A major purpose of a database system is to provide users with an abstract view of the data.

That is, the system hides certain details of how the data are stored and maintained.

The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:



Physical level:

The physical level is the *lowest level of data abstraction* in a database system. It deals with how the data is actually stored in the computer system, including the file structures, indexing methods, and storage mechanisms.

This level involves complex data structures such as B+ trees, hash tables, and data blocks, which are used to manage and organize the data efficiently on the storage device.

Although this level is crucial for performance optimization, users and even most database administrators do not interact with it directly.

Instead, it is handled by database system developers and storage engineers who ensure that data is stored securely and can be retrieved quickly.

Logical level:

The logical level is the next-higher level of abstraction, and *it describes what data is stored in the database and what relationships exist among those data.*

It represents the entire database using simple structures such as tables, attributes (columns), and tuples (rows), without showing how these structures are implemented at the physical level.

This level is primarily used by database administrators who design and maintain the overall database schema. One of the main advantages of the logical level is **physical data independence**, meaning changes to the physical storage (like how data is indexed) do not affect the logical structure of the database.

Users interacting with this level focus on the structure and organization of data, not its storage.

View level:

The view level is the highest level of abstraction and presents only a part of the entire database to the users. *It is designed to simplify user interaction with the system by allowing access to only the relevant data needed for specific tasks.*

For example, a student may see only their own grades, while a teacher may see the grades of all students but not their personal details.

This level helps in hiding the complexity and details of the entire database, providing a cleaner and more user-friendly interface.

Additionally, the view level enhances security by restricting access to sensitive data and allowing different users to have customized views of the same database.

College Student Database System.

1. **Physical Level (How data is stored):** At this level, data about students is stored as files in binary format on a hard disk.

For example, your name, roll number, and marks are stored as bits and bytes using structures like records, blocks, indexes, etc.

The string John may be stored in binary as 01001010 01101111 01101000 01101110 and database knows how to convert that into readable format.

2. Logical Level (What data is stored and how it's related)

We define the structure of the database using tables and relationships.

This level shows *what* information is stored (students' names, roll numbers, department, marks) and how the tables are linked (maybe another table for teachers or subjects). You don't see how the data is actually saved in files.

Roll_No	Name	Dept	Marks
101	John	CS	85
102	Priya	IT	90

3. View Level (What users see) :

Different users see only what they need, not the entire database.

- A **student** logging into a portal sees only their own data
- Name: John, Marks: 85
- A **teacher** sees the names and marks of students in her subject but not their contact info or passwords.
- An **admin** has a different view showing fees, attendance, and more details.

- 2) Explain the concept of candidate key and primary key, foreign key. Identify above listed key for the following schema: Person (driver_id, name, address, contactno) Car(licence, model, year) Owns (driver_id, licence)

1. Super Key :

A super key is any set of attributes that uniquely identifies a tuple (row) in a relation (table).

It can have additional attributes that are not necessary for unique identification.

Emp_ID	Emp_Name	DOB	Gender	Dept_No
E101	Ramkumar	15-JUL-1986	M	2
E103	Ramesh	04-MAY-1989	M	1
E104	Stephen	29-OCT-1987	M	1
E102	Nirmal	23-JAN-1980	M	3
E105	Laxmi	20-MAY-1988	F	4
E107	Rani	23-JAN-1980	F	4
E106	Ramesh	12-MAR-1979	M	2

SuperKey- {Emp_ID}

{Emp_ID,Emp_Name},{Emp_ID,DOB},

{Emp_Name,DOB},

{Emp_Name, DOB, Gender},...

2. Candidate Key :

Definition: A candidate key is a minimal super key; no attribute can be removed without losing unique identification.

Can be more than one in a table.

None of their attributes is redundant.

Example:

If both student_id and email uniquely identify a student, both are candidate keys.

SuperKey- {Emp_ID}
 {Emp_ID,Emp_Name},{Emp_ID,DOB},
 {Emp_Name,DOB},
 {Emp_Name, DOB, Gender},...

Candidate Key – {Emp_ID}
 {Emp_Name, DOB}

3. Primary Key: The primary key is a special column (or a set of columns) in a database table chosen to uniquely identify each record (row) in that table.

It is selected from among the candidate keys by the database designer as the main way to identify entities in an entity set.

Uniqueness: Every value in the primary key must be unique. No two rows can have the same primary key value.

Not Null: Primary key columns cannot have NULL values. Every record must have a value for the primary key field(s).

Example : {Emp_Id};

4 . Foreign Key:

A foreign key is an attribute (or set of attributes) in one table (relation), say r1, that refers to the primary key of another table, say r2.

This creates a relationship between the two tables, ensuring that the value in the foreign key column of r1 must exist as a primary key value in r2.

Roles of Relations

r1 (referencing relation): The table that contains the foreign key.

r2 (referenced relation): The table whose primary key is being referred to.

Example:

Imagine you have two tables:

Students and **Schools**.

Schools table has a column called **School_ID** that gives each school a unique number.
(This is the **primary key of the Schools table**.)

Students table has a column called **School_ID** as well, to show which school each student goes to.

The **School_ID** in the **Students table** is called a foreign key, because it "points to" the main **School_ID** in the **Schools table**.

2. Schema Given

1. Person(driver_id , name, address, contactno)
2. Car(licence , model, year)
3. Owns(driver_id , licence)

Person

- Candidate Key: {driver_id} , {contactno} (if each person has a unique contact number)
- Primary Key: driver_id (chosen to uniquely identify a person)
- Foreign Key: None (this is an independent table)

Car

- Candidate Key: {licence} (license numbers are unique)
- Primary Key: licence
- Foreign Key: None (independent table)

Owns

- Candidate Key: {driver_id, licence} (both together uniquely identify ownership record)
- Primary Key: {driver_id, licence} (composite key)
- Foreign Keys:
 - driver_id → Person(driver_id)
 - licence → Car(licence)

3) Draw architecture of DBMS system and explain function of following components:

i) Storage manager

ii) Query Processor

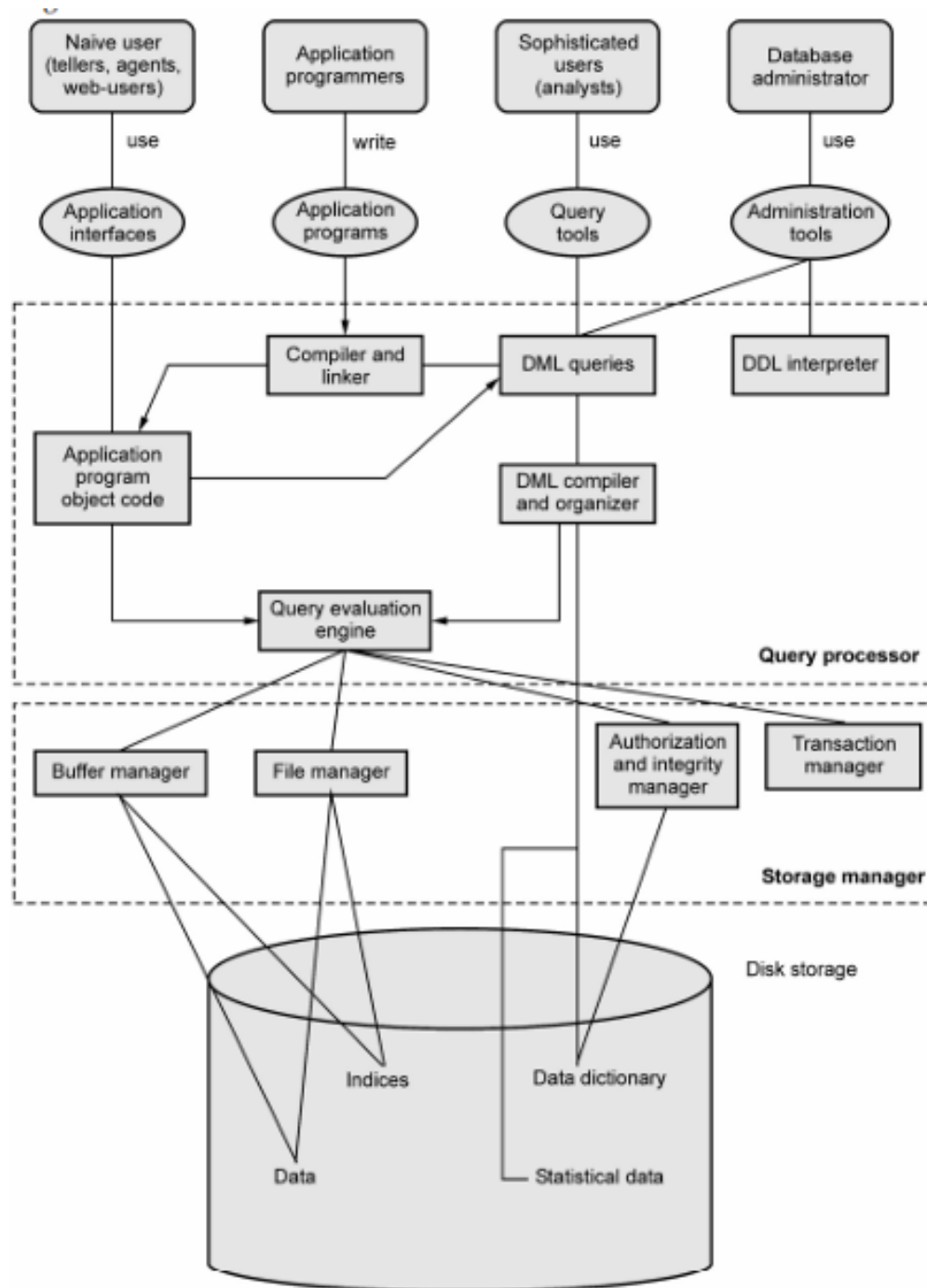


Fig. 1.8.2 Architecture of database

Query processor :

- The interactive query processor helps the database system to simplify and facilitate access to data. It consists of DDL interpreter, DML compiler and query evaluation engine.
- With the following components of query processor, various functionalities are performed -
 - i) **DDL interpreter** : This is basically a translator which interprets the DDL statements in data dictionaries.
 - ii) **DML compiler** : It translates DML statements query language into an evaluation plan. This plan consists of the instructions which query evaluation engine understands.
 - iii) **Query evaluation engine** : It executes the low-level instructions generated by the DML compiler.
- When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query. It is evaluated by **query evaluation engine**.

Storage Manager :

- Storage manager is the component of database system that provides interface between the low level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for storing, retrieving, and updating data in the database. The storage manager components include -
 - i) **Authorization and integrity manager** : Validates the users who want to access the data and tests for integrity constraints.
 - ii) **Transaction manager** : Ensures that the database remains in consistent despite of system failures and concurrent transaction execution proceeds without conflicting.
 - iii) **File manager** : Manages allocation of space on disk storage and representation of the information on disk.

- iv) **Buffer manager** : Manages the fetching of data from disk storage into main memory. The buffer manager also decides what data to cache in main memory. Buffer manager is a crucial part of database system.
- o Storage manager implements several data structures such as -
 - i) **Data files** : Used for storing database itself.
 - ii) **Data dictionary** : Used for storing metadata, particularly schema of database.
 - iii) **Indices** : Indices are used to provide fast access to data items present in the database

- 4) List the main characteristics of the database approach and explain how it differs from the traditional file system.

Main Characteristics of the Database Approach

1. Self-describing nature of the database system
 - A database contains not only the data but also the **metadata** (data about data) describing structure, constraints, and relationships.
2. Insulation between programs and data
 - Data is stored independently from application programs — changes in data structure do not require rewriting all programs.
3. Data abstraction
 - Users interact with **abstract views** of the data without worrying about internal storage details.
4. Support for multiple views of data
 - Different users can see different **subsets** or **representations** of the same database.
5. Sharing of data and multi-user transaction processing
 - Many users can access the same database simultaneously with **concurrency control** and **transaction management**.
6. Data integrity and security
 - Enforces constraints, validation rules, and access control.
7. Data independence
 - **Logical independence**: Change in schema does not affect applications.
 - **Physical independence**: Change in storage details does not affect logical schema.
8. Centralized management
 - A DBMS centrally manages storage, access, backup, recovery, and security.

Sr. No	DBMS	FILE SYSTEM
1	A software framework is DBMS or Database Management System. It is used to access, build and maintain databases.	A file system is a program for handling and organizing the files into a storage medium. It governs the collection and retrieval of data.
2	DBMS offers an abstract image of the data	Details on data representation and data management are supported by the file system.
3	DBMS performs best since a wide range of tools is available to store and retrieve data.	Data can not be stored and recovered in a file system easily.

4	Database Management System stores records, specified restrictions, and interrelationships.	You can save data as isolated data and as entities via File System.
5	It provides an abstract view of data which hides the details	It provides the details of the storage of the data and data representation
6	DBMS secures the data from the system failures by providing a crash recovery mechanism	The file system does not provide a crash mechanism, hence the content of the file will be lost if the system crashes suddenly.
7	It provides higher security	It provides less security
8	It offers a concurrency facility.	It does not offer a concurrency facility.

5) Compare DBMS and File processing system in terms of data isolation, data redundancy, data inconsistency, data integrity.

Aspect	DBMS	File Processing System
Data Isolation	Data is stored in a centralized database, easily accessible through queries; minimal isolation issues.	Data is scattered in separate files, making it difficult to access and integrate.
Data Redundancy	Reduced redundancy due to shared database and controlled storage.	High redundancy because each application maintains its own data files.
Data Inconsistency	Low — updates in one place are reflected everywhere due to central control.	High — duplicate copies may not all be updated, leading to mismatches.
Data Integrity	Integrity constraints (e.g., primary key, foreign key, check) are enforced by DBMS automatically.	No built-in integrity checks; integrity must be handled manually by application programs.

- 6) A weak entity set can always be made into a strong entity set; by adding to its attributes the primary key attributes of its identifying entity set. Outline what sort of redundancy will result if we do so while converting to tables

Key Idea

- A weak entity set does not have a primary key of its own.
- It is identified using:
 1. Its own *partial key* (if any), and
 2. The primary key of its identifying (owner) entity.
- In the E-R model, we normally keep it weak to avoid redundancy.

When converting to a strong entity set

If we "force" a weak entity into a strong one by adding the primary key attributes of its identifying entity to its own attributes, we:

- Give it a composite primary key containing the owner's key.
- Now the identifying entity's attributes are **repeated** for every weak entity occurrence.

Type of Redundancy

The redundancy is repetition of owner's key (and possibly other attributes) in multiple tuples of the converted table.

Example

Original Weak Entity:

- Dependent(Dependent_Name, Age, *linked to* Employee via Emp_ID)

Normal Table Structure (with weak entity)

- Employee(Emp_ID, Name, Dept)
- Dependent(Emp_ID, Dependent_Name, Age)
 - Here, Emp_ID only stored as a foreign key.

If made strong:

- Dependent would now have attributes: (Emp_ID, Name, Dept, Dependent_Name, Age)
 - Problem: **Name** and **Dept** of the employee are repeated for *every dependent* of that employee.

Redundancy Consequences

1. Storage waste – same owner's data stored multiple times.
2. Update anomalies – if owner's attribute changes, must update in many rows.
3. Insertion anomalies – cannot add a dependent without repeating all owner details.
4. Deletion anomalies – deleting last dependent could accidentally lose owner's data.

7) What is Specialization and generalization in Extended E-R diagram? What is the difference between them? Why do we not display the difference in schema diagram?

1. Specialization (Top-Down Approach)

- **Definition:**
The process of **dividing** a higher-level entity into **lower-level sub-entities** based on some distinguishing characteristics.
- **Goal:** Identify more specific entities from a general one.
- **Example:**
 - *Entity:* Employee
 - *Specialization:* → Clerk, Manager, Engineer
- **When used:**
When certain attributes or relationships apply only to some subgroups.

2. Generalization (Bottom-Up Approach)

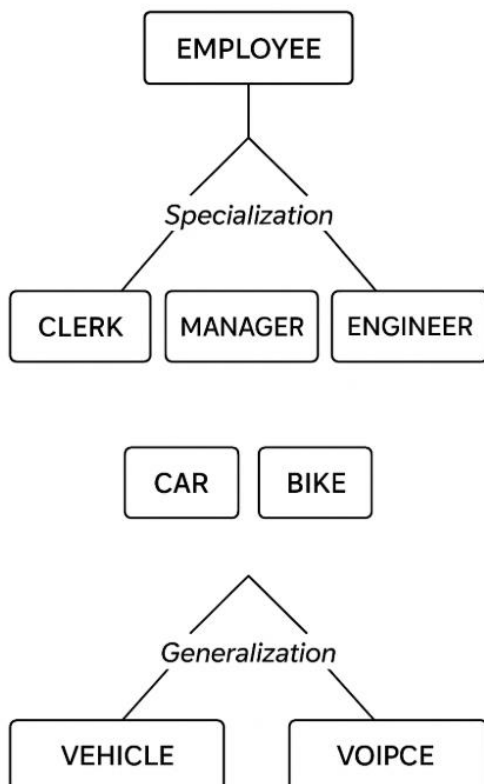
- **Definition:**
The process of **combining** two or more lower-level entities into a **higher-level general entity** that shares common attributes.
- **Goal:** Abstract common features into a single super-entity.
- **Example:**
 - *Entities:* Car, Bike
 - *Generalization:* → Vehicle (common attributes: reg_no, owner, color)
- **When used:**
When different entities have similar attributes/relationships.

3. Difference Between Specialization & Generalization

Aspect	Specialization	Generalization
Approach	Top-down	Bottom-up
Purpose	To identify specific sub-entities from a general entity	To merge similar entities into one general entity
Focus	Differences between subgroups	Common features among entities
Example	Employee → Clerk, Manager	Car + Bike → Vehicle
Entity Movement	One entity becomes multiple entities	Multiple entities become one entity

4. Why the difference is not shown in Schema Diagram

- In the schema diagram, both specialization and generalization are represented in the same way — as a superclass/subclass relationship.
- The diagram only shows the "is-a" hierarchy; it doesn't matter whether it was formed by specialization or generalization because:
 1. The final structure (entities + their relationships) looks the same.
 2. DBMS design focuses on what exists, not how it was derived.
 3. At implementation level, both result in the same table structures.



UNIT 2



8) What is view and how to create it? Can you update view? If yes, how? If not, why not?

1. What is a View?

- A **view** is a **virtual table** in SQL.
- It does **not store data** itself — it displays data stored in underlying tables.
- It is created by saving a SQL `SELECT` query with a name.
- Benefits:
 - Simplifies complex queries
 - Provides security by showing only certain columns/rows
 - Gives a customized presentation of data

2. How to Create a View



sql

 Copy  Edit

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

sql

 Copy  Edit

```
CREATE VIEW Active_Customers AS
SELECT Customer_ID, Name
FROM Customers
WHERE Status = 'Active';
```

3. Can You Update a View?

Yes, but only if the view is "updatable"

A view is updatable when:

- It is based on **one table** (not multiple joined tables).
- It does not use:
 - `DISTINCT`
 - Aggregate functions (`SUM`, `AVG`, etc.)
 - `GROUP BY` or `HAVING`
 - Set operations (`UNION`, `INTERSECT`, etc.)
- All **NOT NULL** columns without defaults are included in the view.

Example of updating a view:

sql

 Copy  Edit

```
UPDATE Active_Customers
SET Name = 'John Doe'
WHERE Customer_ID = 101;
```

This will update the **underlying table**.

When You Cannot Update a View

If the view is:

- Based on **multiple tables** (JOINS)
- Contains **aggregated data**
- Uses calculated columns

Reason:

The DBMS cannot determine **exactly** how the change should be reflected in the original tables.

9) Defined stored procedure. Explain the creating and calling stored procedure with example.

1. Definition of Stored Procedure

- A **Stored Procedure** is a **precompiled set of SQL statements** stored in the database.
- It can take **parameters**, perform operations (like SELECT, INSERT, UPDATE, DELETE), and return results.
- Benefits:
 - Faster execution (precompiled)
 - Reusable
 - Centralized business logic
 - Improves security (users can execute without direct table access)

2. Creating a Stored Procedure

Syntax (MySQL style):

```
sql                                                                    Copy Edit

DELIMITER $$

CREATE PROCEDURE procedure_name (parameters)
BEGIN
    -- SQL statements
END$$

DELIMITER ;
```

Example:

```
sql                                                                    Copy Edit

DELIMITER $$

CREATE PROCEDURE GetCustomerDetails(IN cust_id INT)
BEGIN
    SELECT Customer_ID, Name, Address
    FROM Customers
    WHERE Customer_ID = cust_id;
END$$



DELIMITER ;
```

Here:

- `IN cust_id INT` = Input parameter.
- The procedure fetches customer details based on the ID.

3. Calling a Stored Procedure

sql



 Copy  Edit

```
CALL GetCustomerDetails(101);
```

This will execute the procedure and return the details of the customer with ID `101`.

4. Example with Output Parameter

sql

 Copy  Edit

```
DELIMITER $$
```

```
CREATE PROCEDURE GetCustomerCount(OUT total INT)
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO total
```

```
    FROM Customers;
```

```
END$$
```

```
DELIMITER ;
```

```
CALL GetCustomerCount(@cnt);
```

```
SELECT @cnt;
```

Here:

- `OUT total INT` returns a value to the caller.
- First `CALL` runs the procedure.
- `SELECT @cnt` shows the returned value.



- 10) Consider following schema. Student_fee_details (rollno, name, fee_deposited, date) [5] Write a trigger to preserve old values of student fee details before updating in the table.

1. What is a Trigger?

Definition:

A **trigger** is a special stored program in the database that **automatically executes** when a specified event occurs on a table or view.

Events that can activate triggers:

- INSERT
- UPDATE
- DELETE

Key Points:

- Triggers run **automatically** — you do not call them manually.
- Can be **BEFORE** or **AFTER** the event.
- Often used for:
 - Maintaining audit logs
 - Enforcing business rules
 - Preventing invalid operations

2. Example — Trigger to Preserve Old Values

You want to **store old values** from `Student_fee_details` **before updating**.

Step 1: Create a backup/audit table

sql

Copy Edit

```
CREATE TABLE Student_fee_details_backup (  
  rollno INT,  
  name VARCHAR(50),  
  fee_deposited DECIMAL(10,2),  
  date DATE  
);
```

Step 2: Create the trigger

sql

Copy Edit

```
DELIMITER $$
```

```
CREATE TRIGGER backup_before_update
```

```
BEFORE UPDATE ON Student_fee_details
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO Student_fee_details_backup
```

```
    VALUES (OLD.rollno, OLD.name, OLD.fee_deposited, OLD.date);
```

```
END$$
```

```
DELIMITER ;
```

Explanation:

- **BEFORE UPDATE** → Trigger runs before the update happens.
- **FOR EACH ROW** → Works on each row being updated.
- **OLD.column_name** → Refers to the existing value before update.
- We insert the old values into `Student_fee_details_backup` for record keeping.

11) Write SQL queries for the following:

- Find shipment information (SNO, Sname, PNO, Pname, QTY) for those having quantity less than 157.
- List SNO, Sname, PNO, Pname for those suppliers who made shipments of parts whose quantity is larger than the average quantity.
- Find aggregate quantity of PNO 1692 of color green for which shipments were made by supplier number who is residing in Mumbai.

Given Schemas

text

Copy Edit

```
Supplier(SNO, Sname, Status, City)
Parts (PNO, Pname, Color, Weight, City)
Shipments(SNO, PNO, QTY)
```

i) Find shipment information (SNO, Sname, PNO, Pname, QTY) for those having quantity less than 157

sql

Copy Edit

```
SELECT s.SNO, s.Sname, p.PNO, p.Pname, sh.QTY
FROM Supplier s
JOIN Shipments sh ON s.SNO = sh.SNO
JOIN Parts p ON sh.PNO = p.PNO
WHERE sh.QTY < 157;
```

- JOIN connects suppliers → shipments → parts.
- WHERE sh.QTY < 157 filters shipments.

ii) List SNO, Sname, PNO, Pname for suppliers who shipped parts with quantity larger than the average quantity

sql

Copy Edit

```
SELECT DISTINCT s.SNO, s.Sname, p.PNO, p.Pname
FROM Supplier s
JOIN Shipments sh ON s.SNO = sh.SNO
JOIN Parts p ON sh.PNO = p.PNO
WHERE sh.QTY > (SELECT AVG(QTY) FROM Shipments);
```

- AVG(QTY) finds the average quantity of all shipments.
- DISTINCT avoids duplicate rows if same supplier ships multiple times.

iii) Find aggregate quantity of PNO 1692 of color green for shipments made by suppliers from Mumbai

sql

Copy Edit

```
SELECT SUM(sh.QTY) AS Total_Quantity
FROM Supplier s
JOIN Shipments sh ON s.SNO = sh.SNO
JOIN Parts p ON sh.PNO = p.PNO
WHERE p.PNO = 1692
      AND p.Color = 'Green'
      AND s.City = 'Mumbai';
```

- **SUM(sh.QTY)** calculates the total.
- Filters by:
 - `p.PNO = 1692`
 - `p.Color = 'Green'`
 - Supplier's city is `'Mumbai'`

12) What is an index? What are the advantages and disadvantages of using index on a table?

1. What is an Index?

- An **index** in a database is a **data structure** (like a book's index) that speeds up the retrieval of rows from a table.
- It works like a **pointer** to data — instead of scanning the whole table, the DBMS uses the index to find rows quickly.
- Usually implemented using **B-trees** or **hash tables**.

Example:

sql

Copy Edit

```
CREATE INDEX idx_customer_name  
ON Customers (Name);
```

This creates an index on the `Name` column of `Customers` to speed up searches.

2. Advantages of Index

1. **Faster Data Retrieval** – Improves query performance for `SELECT` operations, especially on large tables.
2. **Quick Sorting** – Helps with `ORDER BY` and `GROUP BY` queries.
3. **Efficient Searching** – Greatly speeds up `WHERE` conditions.
4. **Reduced Disk I/O** – Avoids scanning the full table.

3. Disadvantages of Index

1. **Extra Storage** – Indexes require additional disk space.
2. **Slower Data Modification** – `INSERT`, `UPDATE`, and `DELETE` become slower because the index must be updated each time data changes.
3. **Maintenance Overhead** – More indexes = more work for the DBMS during data changes.
4. **Not Useful for Small Tables** – Overhead can outweigh performance gains.

13) Discuss various types of triggers

1. Based on Triggering Event

These are categorized by the **SQL operation** that activates them:

Trigger Type	Description	Example Use
INSERT Trigger	Fires when a new row is inserted into the table.	Automatically log new user registrations.
UPDATE Trigger	Fires when a row is updated.	Store old values before changes for audit.
DELETE Trigger	Fires when a row is deleted.	Keep a backup of deleted records.


2. Based on Trigger Timing

Defines **when** the trigger fires in relation to the event:

Type	When it Executes	Example Use
BEFORE Trigger	Runs before the event executes.	Validate data before inserting.
AFTER Trigger	Runs after the event executes.	Log changes after they are made.
INSTEAD OF Trigger	Replaces the triggering action (mainly for views).	Allow updates on a view by redirecting changes to underlying tables.

3. Based on Execution Scope

Determines how many times the trigger fires:

Type	Description	Example Use	
ROW-level Trigger	Fires once for each row affected.	Keep audit trail for every updated row.	
STATEMENT-level Trigger	Fires once per SQL statement , regardless of rows affected.	Maintain summary counters.	

14) Consider the following schemas
Emp(Emp_no, Emp_name, Dept_no)
Dept(Dept_no, Dept_name)
Address(Dept_name, Dept_location)
Write SQL queries for the following

- i) Display the location of department where employee 'Ram' is working.
- ii) Create a view to store total no of employees working in each department in ascending order.
- iii) Find the name of the department in which no employee is working.

Given:



scss

 Copy  Edit

```
Emp(Emp_no, Emp_name, Dept_no)
Dept(Dept_no, Dept_name)
Address(Dept_name, Dept_location)
```

i) Display the location of department where employee 'Ram' is working:

sql

 Copy  Edit

```
SELECT A.Dept_location
FROM Emp E
JOIN Dept D ON E.Dept_no = D.Dept_no
JOIN Address A ON D.Dept_name = A.Dept_name
WHERE E.Emp_name = 'Ram';
```

ii) Create a view to store total no of employees working in each department in ascending order:



sql

 Copy  Edit

```
CREATE VIEW Dept_Emp_Count AS
SELECT D.Dept_name, COUNT(E.Emp_no) AS Total_Employees
FROM Dept D
LEFT JOIN Emp E ON D.Dept_no = E.Dept_no
GROUP BY D.Dept_name
ORDER BY Total_Employees ASC;
```

iii) Find the name of the department in which no employee is working:

sql

 Copy  Edit

```
SELECT Dept_name
FROM Dept
WHERE Dept_no NOT IN (SELECT Dept_no FROM Emp);
```

15) Explain the concept of Referential and Entity Integrity constraint with example.

1. Entity Integrity Constraint

- **Meaning:** Ensures that every table (relation) has a **primary key** and that this key **cannot have NULL values**.
- **Purpose:** Guarantees that each row in a table is uniquely identifiable.
- **Rule:** Primary key attributes must be **unique** and **NOT NULL**.

Example:

sql

Copy Edit

```
CREATE TABLE Student (  
    RollNo INT PRIMARY KEY,  
    Name VARCHAR(50)  
);
```

- Here, `RollNo` is the primary key → cannot be NULL and must be unique.

2. Referential Integrity Constraint

- **Meaning:** Ensures that the value of a **foreign key** in one table matches an existing value of a **primary key** in another table.
- **Purpose:** Maintains consistency between related tables.
- **Rule:** A foreign key must either be NULL or match an existing primary key value in the referenced table.

Example:

sql

Copy Edit

```
CREATE TABLE Department (  
    DeptID INT PRIMARY KEY,  
    DeptName VARCHAR(50)  
);  
  
CREATE TABLE Employee (  
    EmpID INT PRIMARY KEY,  
    EmpName VARCHAR(50),  
    DeptID INT,  
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)  
);
```

- Here, `DeptID` in `Employee` must exist in `Department` or be NULL.
- If `DeptID = 5` does not exist in `Department`, it cannot be inserted into `Employee`.

- 16) Write a PL/SQL block of code which accepts the rollno from user. The attendance of rollno entered by user will be checked in student_attendance(RollNo, Attendance) table and display on the screen.

```
SET SERVEROUTPUT ON;
DECLARE
    v_rollno      student_attendance.RollNo%TYPE;
    v_attendance  student_attendance.Attendance%TYPE;
BEGIN
    -- Accept roll number from user
    v_rollno := &Enter_RollNo;

    -- Retrieve attendance
    SELECT Attendance
    INTO v_attendance
    FROM student_attendance
    WHERE RollNo = v_rollno;

    -- Display result
    DBMS_OUTPUT.PUT_LINE('Roll No: ' || v_rollno || ' | Attendance: ' || v_attendance || '%');

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No record found for Roll No: ' || v_rollno);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

- 17) What is the importance of creating constraints on the table? Explain with example any 4 constraints that can be specified when a database table is created.

Importance of Creating Constraints on a Table

Constraints are rules applied on table columns to maintain **accuracy**, **consistency**, and **reliability** of data in a database. They ensure that only valid data is entered, prevent accidental data corruption, and help enforce business rules directly at the database level rather than relying only on application code.

Without constraints, wrong, duplicate, or incomplete data could be stored, leading to incorrect results and potential system failures.

Four Common Constraints with Examples

1. PRIMARY KEY

- Ensures that each record in a table is **unique** and **not NULL**.
- A table can have only **one** primary key (single or composite).

Example:

sql

Copy Edit

```
CREATE TABLE Students (  
    RollNo INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Age INT  
);
```

Here, `RollNo` uniquely identifies each student and cannot be NULL.

2. FOREIGN KEY

- Maintains **referential integrity** between two tables.
- Ensures a value in one table matches an existing value in another table.

Example:

sql

Copy Edit

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

Here, `CustomerID` in `Orders` must exist in the `Customers` table.

3. UNIQUE

- Ensures all values in a column are **different** (but allows **one NULL** by default).

Example:

sql

CopyEdit

```
CREATE TABLE Employees (  
    EmpID INT PRIMARY KEY,  
    Email VARCHAR(100) UNIQUE  
);
```

No two employees can have the same `Email`.

4. CHECK

- Ensures that column values meet a specific condition.

Example:

sql

CopyEdit

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    Price DECIMAL(10,2) CHECK (Price > 0)  
);
```

Here, `Price` must always be greater than zero.

✔ Summary Table:

Constraint	Purpose	NULL Allowed?	Example Use
PRIMARY KEY	Uniquely identifies rows	✗	<code>RollNo</code> in Students
FOREIGN KEY	Enforces relationship between tables	✔	<code>CustomerID</code> in Orders
UNIQUE	Ensures all values are unique	✔ (1 allowed)	<code>Email</code> in Employees
CHECK	Restricts values to meet a condition	✔	<code>Price > 0</code> in Products

18) Consider the following schema:

Hotels(hotel_no, hotel_name, city)

Rooms(Room_no, hotel_no, price, type)

Write a PL/SQL procedure to list the price and type of all rooms at the hotel 'TAJ'.

```
sql

CREATE OR REPLACE PROCEDURE List_Taj_Rooms AS
BEGIN
  FOR rec IN (
    SELECT r.price, r.type
    FROM Hotels h
    JOIN Rooms r ON h.hotel_no = r.hotel_no
    WHERE h.hotel_name = 'TAJ'
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Price: ' || rec.price || ', Type: ' || rec.type);
  END LOOP;
END;
/
```

```
SET SERVEROUTPUT ON;
EXEC List_Taj_Rooms;
```



19) Consider the following relation schema
MOVIES(Mov_Id, Mov_Title, Mov_Year, Dir_Id)
DIRECTOR(Dir_Id, Dir_Name)
RATING(MOV_Id, Rev_Stars)

Write the SQL queries for the following:

- i) List the title of all the movies directed by 'RAJ KAPOOR'.
- ii) Find the name of movies and number of stars for each movie. Sort the results on movies title and from higher stars to least stars.
- iii) Assign the rating of all movies directed by 'Steven Spielberg' to 9.

i) List the title of all the movies directed by 'RAJ KAPOOR'



sql

 Copy  Edit

```
SELECT m.Mov_Title
FROM MOVIES m
JOIN DIRECTOR d ON m.Dir_Id = d.Dir_Id
WHERE d.Dir_Name = 'RAJ KAPOOR';
```

ii) Find the name of movies and number of stars for each movie. Sort the results on movie title and from higher stars to least stars.


sql

 Copy  Edit

```
SELECT m.Mov_Title, r.Rev_Stars
FROM MOVIES m
JOIN RATING r ON m.Mov_Id = r.Mov_Id
ORDER BY m.Mov_Title ASC, r.Rev_Stars DESC;
```

iii) Assign the rating of all movies directed by 'Steven Spielberg' to 9.

sql

 Copy  Edit



```
UPDATE RATING
SET Rev_Stars = 9
WHERE Mov_Id IN (
    SELECT m.Mov_Id
    FROM MOVIES m
    JOIN DIRECTOR d ON m.Dir_Id = d.Dir_Id
    WHERE d.Dir_Name = 'Steven Spielberg'
);
```

20) What is synonym? How to create and use synonym in SQL

A synonym is an alternative name for a database object such as a table, view, sequence, procedure, etc. It is used to simplify object names or hide the schema details from users.

Creating a Synonym



sql

 Copy  Edit

```
CREATE SYNONYM synonym_name FOR original_object_name;
```

Example:

sql

 Copy  Edit

```
CREATE SYNONYM emp FOR hr.employees;
```

Here, `emp` is a synonym for the `hr.employees` table.

Using a Synonym

Once created, you can use the synonym instead of the original object name:



sql

 Copy  Edit

```
SELECT * FROM emp;
```

This works the same as:

sql

 Copy  Edit

```
SELECT * FROM hr.employees;
```

Benefits

- Simplifies SQL statements
- Provides location transparency
- Hides schema details from users

All The Best

- Karan Salunkhe 😊