Module 4 – Introduction to DBMS

Theory Questions:

1. What is SQL, and why is it essential in database management?

Ans . **SQL (Structured Query Language)** is a standardized programming language used to manage and manipulate relational databases. It is essential in database management for several key reasons:

**Core Functions of SQL:**

* **Data Querying**: Retrieve specific data using SELECT statements.
* **Data Manipulation**: Add, update, or delete data using INSERT, UPDATE, and DELETE.
* **Data Definition**: Create or modify database structures using CREATE, ALTER, and DROP.
* **Data Control**: Manage user access and permissions with GRANT and REVOKE.

**Why SQL is Essential:**

1. **Universal Standard**: Widely adopted across relational database systems like MySQL, PostgreSQL, SQL Server, and Oracle.
2. **Efficient Data Handling**: Allows for complex queries and joins to extract meaningful insights from large datasets.
3. **Data Integrity**: Supports constraints, transactions, and referential integrity to ensure consistent and accurate data.
4. **User & Access Management**: Enables fine-grained control over who can view or manipulate data.
5. **Automation & Integration**: Easily integrates with various programming languages and tools, automating data workflows.

In summary, SQL is the backbone of relational database operations, enabling users and applications to efficiently access, modify, and manage structured data.

1. Explain the difference between DBMS and RDBMS.

Ans. **Difference Between DBMS and RDBMS**

| **Feature** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Definition** | Software for storing and managing data in databases. | A type of DBMS that stores data in **tables (relations)** with defined relationships. |
| **Data Storage** | Data can be stored in files, trees, or other formats. | Data is stored **in tabular format (rows and columns)**. |
| **Data Relationships** | Does **not enforce relationships** between data. | Enforces relationships using **foreign keys** and **primary keys**. |
| **Normalization** | May not support normalization, leading to data redundancy. | Follows **normalization rules** to eliminate redundancy. |
| **Integrity Constraints** | Integrity constraints (like uniqueness, referential integrity) are **not strictly enforced**. | **Supports and enforces** constraints such as primary key, foreign key, unique, and not null. |
| **Examples** | File systems, XML, Microsoft Access (basic usage) | MySQL, PostgreSQL, Oracle, Microsoft SQL Server |
| **Multi-user Environment** | Limited or no support for concurrent users. | Designed for **multi-user** access with proper concurrency controls. |
| **Security** | Basic security features. | **Advanced security features** like role-based access control. |

1. Describe the role of SQL in managing relational databases.

Ans. SQL (Structured Query Language) plays a **central role** in managing relational databases. It provides a standardized way to interact with data stored in **tables**—the fundamental structure of relational databases.

### Key Roles of SQL:

#### 1. **Data Querying**

* SQL allows users to retrieve specific data using the SELECT statement.
* Supports filtering (WHERE), sorting (ORDER BY), and joining multiple tables (JOIN).
* Example:

sql

CopyEdit

SELECT name, salary FROM employees WHERE department = 'HR';

#### 2. **Data Manipulation (DML)**

* Enables inserting, updating, and deleting data.
* Common DML commands: INSERT, UPDATE, DELETE.
* Example:

sql

CopyEdit

UPDATE products SET price = price \* 1.10 WHERE category = 'Electronics';

#### 3. **Data Definition (DDL)**

* Used to define and modify the structure of database objects like tables, indexes, and schemas.
* Commands: CREATE, ALTER, DROP.
* Example:

sql

CopyEdit

CREATE TABLE students (

id INT PRIMARY KEY,

name VARCHAR(100),

age INT

);

#### 4. **Data Control (DCL)**

* Manages access permissions and security.
* Commands: GRANT, REVOKE.
* Example:

sql

CopyEdit

GRANT SELECT ON employees TO hr\_user;

#### 5. **Transaction Control (TCL)**

* Ensures data integrity and consistency during operations.
* Commands: COMMIT, ROLLBACK, SAVEPOINT.
* Example:

sql

CopyEdit

BEGIN;

UPDATE accounts SET balance = balance - 500 WHERE id = 1;

UPDATE accounts SET balance = balance + 500 WHERE id = 2;

COMMIT;

Top of Form

Bottom of Form

1. What are the key features of SQL?

Ans SQL is a powerful and standardized language used to interact with relational databases. Here are its **key features**:

**🔹 1. Data Querying**

* SQL can retrieve specific data from one or more tables.
* The SELECT statement is used to fetch data with filtering (WHERE), sorting (ORDER BY), and grouping (GROUP BY).

**Example:**

sql

CopyEdit

SELECT name, salary FROM employees WHERE department = 'IT';

**🔹 2. Data Manipulation**

* SQL supports modification of data using:
  + INSERT – Add new records
  + UPDATE – Modify existing records
  + DELETE – Remove records

**🔹 3. Data Definition**

* SQL allows creation and modification of database structures like tables and schemas:
  + CREATE, ALTER, DROP for defining and modifying database objects.

**🔹 4. Data Control**

* Manages access to data through:
  + GRANT – Assign privileges
  + REVOKE – Remove privileges

**🔹 5. Transaction Control**

* Ensures reliability and consistency of data operations:
  + BEGIN, COMMIT, ROLLBACK, SAVEPOINT help manage transactions.

**🔹 6. Support for Functions and Clauses**

* SQL includes built-in functions for:
  + **Aggregation**: SUM(), COUNT(), AVG()
  + **String handling**: UPPER(), LOWER(), CONCAT()
  + **Date/time functions** and **mathematical operations**

**🔹 7. Data Integrity & Constraints**

* Enforces rules on data using:
  + PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK

**🔹 8. Standardized Language**

* SQL is ANSI and ISO standard, widely supported across database systems like MySQL, PostgreSQL, Oracle, SQL Server, and SQLite.

**🔹 9. Portability**

* SQL code is generally portable across different relational database systems with minimal changes.

LAB EXERCISES:

1. Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address.

### Ans. ****tep 1: Create the Database****

sql

CopyEdit

CREATE DATABASE school\_db;

✅ This creates a new database named school\_db.

### 🔹 ****Step 2: Use the Database****

sql

CopyEdit

USE school\_db;

✅ This tells SQL to run subsequent commands in the school\_db database.

### 🔹 ****Step 3: Create the**** students ****Table****

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100),

age INT,

class VARCHAR(10),

address VARCHAR(255)

);

1. Insert five records into the students table and retrieve all records using the SELECT statement.

Ans. INSERT INTO students (student\_id, student\_name, age, class, address) VALUES

(1, 'Alice Johnson', 14, '8A', '123 Maple St'),

(2, 'Brian Lee', 15, '9B', '456 Oak Ave'),

(3, 'Catherine Smith', 13, '7C', '789 Pine Rd'),

(4, 'David Brown', 14, '8A', '321 Cedar Blvd'),

(5, 'Eva Green', 16, '10A', '654 Birch Ln');

2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?

### Ans. Basic Components of SQL Syntax

SQL syntax consists of specific elements and rules that form the structure of SQL statements. Understanding these components helps in writing valid and effective SQL queries.

**🔹 1. Statements/Commands**

SQL is made up of various **commands** grouped into categories:

* **DML (Data Manipulation Language)**: SELECT, INSERT, UPDATE, DELETE
* **DDL (Data Definition Language)**: CREATE, ALTER, DROP
* **DCL (Data Control Language)**: GRANT, REVOKE
* **TCL (Transaction Control Language)**: COMMIT, ROLLBACK, SAVEPOINT

**🔹 2. Clauses**

Clauses define the **conditions** or **scope** of a statement.

* SELECT ... FROM
* WHERE: filter conditions
* ORDER BY: sort results
* GROUP BY: aggregate grouping
* HAVING: filter grouped data
* LIMIT: limit number of results (in some systems)

**🔹 3. Expressions**

Expressions combine values, fields, and functions to return a result.

* Arithmetic: salary \* 0.1
* Logical: age > 18 AND class = '10A'
* String: CONCAT(first\_name, ' ', last\_name)

**🔹 4. Predicates**

Predicates return **true/false** and are often used in WHERE, JOIN, or HAVING.

* =, !=, >, <, BETWEEN, IN, LIKE, IS NULL

**🔹 5. Identifiers**

Names of **tables**, **columns**, **databases**, **aliases**.

* Example: students, student\_id, AS s

**🔹 6. Keywords**

Reserved words that have specific meaning in SQL and form the structure of commands.

* Examples: SELECT, FROM, WHERE, JOIN, INSERT, UPDATE, DELETE

**🔹 7. Functions**

Built-in SQL functions to perform operations:

* **Aggregate**: SUM(), AVG(), COUNT(), MAX(), MIN()
* **String**: UPPER(), LOWER(), CONCAT()
* **Date/Time**: NOW(), DATE()

**🔹 8. Comments**

Used to explain SQL code.

* Single-line comment: -- This is a comment
* Multi-line comment:

sql

CopyEdit

/\* This is

a multi-line comment \*/

**🔹 9. Semicolon (;)**

* Used to **terminate** SQL statements (especially when executing multiple statements).

**✅ Example: SQL Syntax in Action**

sql

CopyEdit

SELECT student\_name, age

FROM students

WHERE age > 14

ORDER BY age DESC;

This query:

* Selects student\_name and age
* From the students table
* Where the age is greater than 14
* Orders the result in descending order by age

1. Write the general structure of an SQL SELECT statement.

Ans. SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

[GROUP BY column]

[HAVING group\_condition]

[ORDER BY column [ASC|DESC]]

[LIMIT number]; -- (Optional, supported in some databases like MySQL, PostgreSQL)

1. Explain the role of clauses in SQL statements.

Ans. In SQL, **clauses** are essential components that define **how a query behaves**—they determine what data is retrieved, how it’s filtered, grouped, sorted, and limited. Each clause serves a specific purpose and is used in combination with others to form complete SQL statements.

### 🔹 ****Key Roles of Common SQL Clauses****

| **Clause** | **Role / Function** |
| --- | --- |
| **SELECT** | Specifies which **columns** to retrieve from the table. |
| **FROM** | Identifies the **table(s)** where the data resides. |
| **WHERE** | Filters rows based on **conditions** (e.g., age > 18). |
| **GROUP BY** | Groups rows based on one or more columns to **aggregate** data (e.g., COUNT, SUM). |
| **HAVING** | Filters groups created by GROUP BY based on aggregate values. |
| **ORDER BY** | Sorts the result set by one or more columns, **ascending or descending**. |
| **JOIN** | Combines data from multiple tables based on a **related key column**. |
| **LIMIT / TOP** | Restricts the **number of rows** returned (depending on SQL dialect). |

### 🔸 ****Example Statement with Clauses****

sql

CopyEdit

SELECT department, COUNT(\*) AS total

FROM employees

WHERE status = 'active'

GROUP BY department

HAVING COUNT(\*) > 5

ORDER BY total DESC

LIMIT 5;

#### What Each Clause Does:

* SELECT: Chooses department and count of employees.
* FROM: Queries the employees table.
* WHERE: Filters rows where status is 'active'.
* GROUP BY: Groups results by department.
* HAVING: Only includes groups with more than 5 employees.
* ORDER BY: Sorts the results by the total count, descending.
* LIMIT: Returns only the top 5 results.

1. LAB EXERCISES: • Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

Ans. SELECT student\_name, age

FROM students;

1. Write SQL queries to retrieve all students whose age is greater than 10.

Ans. SELECT \*

FROM students

WHERE age > 10;

SQL Constraints

Theory Questions:

1. What are constraints in SQL? List and explain the different types of constraints.

Ans. **Constraints** in SQL are **rules enforced on data** in a table to ensure its **accuracy, integrity, and consistency**. They help prevent invalid data from being inserted, updated, or deleted in the database.

Constraints are usually defined when creating or altering a table and are automatically enforced by the database.

### 🔹 ****Types of SQL Constraints****

| **Constraint** | **Description** |
| --- | --- |
| **NOT NULL** | Ensures that a column **cannot have a NULL value**. |
| **UNIQUE** | Ensures that **all values in a column are different** (no duplicates). |
| **PRIMARY KEY** | Uniquely identifies each row in a table. It is a combination of NOT NULL and UNIQUE. |
| **FOREIGN KEY** | Creates a relationship between two tables by linking a column to the PRIMARY KEY of another table. |
| **CHECK** | Ensures that **values in a column satisfy a specific condition**. |
| **DEFAULT** | Assigns a **default value** to a column when no value is provided during insert. |
| **INDEX** (not technically a constraint) | Improves the performance of queries on a column but does not enforce rules like other constraints. |

### 🔸 ****Examples****

#### 1. **NOT NULL**

sql

CopyEdit

CREATE TABLE students (

student\_id INT NOT NULL,

student\_name VARCHAR(100) NOT NULL

);

#### 2. **UNIQUE**

sql

CopyEdit

CREATE TABLE employees (

email VARCHAR(100) UNIQUE

);

#### 3. **PRIMARY KEY**

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100)

);

#### 4. **FOREIGN KEY**

sql

CopyEdit

CREATE TABLE enrollments (

enrollment\_id INT PRIMARY KEY,

student\_id INT,

FOREIGN KEY (student\_id) REFERENCES students(student\_id)

);

#### 5. **CHECK**

sql

CopyEdit

CREATE TABLE students (

age INT CHECK (age >= 5)

);

#### 6. **DEFAULT**

sql

CopyEdit

CREATE TABLE students (

city VARCHAR(100) DEFAULT 'Unknown'

1. How do PRIMARY KEY and FOREIGN KEY constraints differ?

### ans. ****1. PRIMARY KEY****

| **Feature** | **Description** |
| --- | --- |
| **Purpose** | Uniquely identifies each **record (row)** in a table. |
| **Uniqueness** | Must contain **unique** values — no duplicates. |
| **NULLs Allowed?** | **No** — cannot contain NULL values. |
| **One per Table?** | Each table can have **only one** primary key (can be a single column or a combination of columns). |
| **Example** | student\_id in a students table. |

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100)

);

### 🔹 ****2. FOREIGN KEY****

| **Feature** | **Description** |
| --- | --- |
| **Purpose** | Creates a **relationship** between two tables by referencing the primary key in another table. |
| **Uniqueness** | **Not required** to be unique. |
| **NULLs Allowed?** | **Yes**, unless explicitly set as NOT NULL. |
| **One per Table?** | A table can have **multiple foreign keys**. |
| **Example** | student\_id in an enrollments table referencing the students table. |

sql

CopyEdit

CREATE TABLE enrollments (

enrollment\_id INT PRIMARY KEY,

student\_id INT,

FOREIGN KEY (student\_id) REFERENCES students(student\_id)

);

1. What is the role of NOT NULL and UNIQUE constraints?

### Ans. ****1.**** NOT NULL ****Constraint****

#### 🔸 **Purpose**:

Ensures that a **column cannot have a NULL value**. It guarantees that a field must always contain a valid (non-null) value when a record is inserted or updated.

#### ✅ **Key Points**:

* Enforces **mandatory data entry**.
* Can be applied to any data type.
* Commonly used for essential fields like names, IDs, emails, etc.

#### 💡 **Example**:

sql

CopyEdit

CREATE TABLE students (

student\_id INT NOT NULL,

student\_name VARCHAR(100) NOT NULL

);

This means every student **must have** an ID and a name — no blanks allowed.

### 🔹 ****2.**** UNIQUE ****Constraint****

#### 🔸 **Purpose**:

Ensures that **all values in a column (or a combination of columns) are different**. It prevents duplicate entries.

#### ✅ **Key Points**:

* Enforces **uniqueness** across the column.
* Allows only **one NULL** (or sometimes multiple depending on DBMS).
* You can define **multiple unique constraints** on different columns in the same table.

#### 💡 **Example**:

sql

CopyEdit

CREATE TABLE users (

user\_id INT PRIMARY KEY,

email VARCHAR(100) UNIQUE

);

LAB EXERCISES: • Lab

1. 1: Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE)

Ans. CREATE TABLE teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE

);

1. Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.

#### Ans. Option 1: Create the students table with the foreign key

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

age INT,

class VARCHAR(50),

address VARCHAR(255),

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id)

);

#### 🔸 Option 2: Add the foreign key to an existing students table

sql

CopyEdit

ALTER TABLE students

ADD teacher\_id INT;

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id);

Main SQL Commands and Sub-commands (DDL)

Theory Questions:

1. Define the SQL Data Definition Language (DDL).

### Ans. ****ey Characteristics of DDL:****

* It deals with the **creation, modification, and deletion** of database structures.
* Changes made using DDL statements are **automatically committed**—they **cannot be rolled back** in most SQL environments.

### 🔸 ****Common DDL Commands****

| **Command** | **Description** |
| --- | --- |
| CREATE | Creates a new database object (e.g., table, view, index). |
| ALTER | Modifies an existing database object (e.g., adding a column to a table). |
| DROP | Deletes an existing database object. |
| TRUNCATE | Removes all rows from a table, **resetting its storage**, but keeps the table structure. |
| RENAME | Renames a database object (like a table or column). |

### 🔍 ****Examples****

#### 🔸 CREATE example:

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100),

age INT

);

#### 🔸 ALTER example:

sql

CopyEdit

ALTER TABLE students ADD email VARCHAR(100);

#### 🔸 DROP example:

sql

CopyEdit

DROP TABLE students;

#### 🔸 TRUNCATE example:

sql

CopyEdit

TRUNCATE TABLE students;

1. Explain the CREATE command and its syntax.

### Ans. ****Purpose of**** CREATE

* Define the **structure** of a new object.
* Allocate **space** and set **constraints** (e.g., NOT NULL, PRIMARY KEY, FOREIGN KEY, etc.).
* Initialize new entities in the database system.

### 🔸 ****1. General Syntax to Create a Table****

sql

CopyEdit

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

);

#### 🔍 Components:

* table\_name: Name of the new table.
* column: Name of each field/column.
* datatype: Data type of the column (e.g., INT, VARCHAR, DATE).
* constraint (optional): Rules applied to columns (e.g., NOT NULL, UNIQUE, PRIMARY KEY).

### 🔹 ****Example: Create a**** students ****Table****

sql

CopyEdit

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

age INT,

class VARCHAR(50),

address VARCHAR(255)

);

### 🔸 ****2. CREATE DATABASE Syntax****

sql

CopyEdit

CREATE DATABASE database\_name;

#### Example:

sql

CopyEdit

CREATE DATABASE school\_db;

### 🔸 ****3. CREATE INDEX Syntax****

sql

CopyEdit

CREATE INDEX index\_name

ON table\_name (column\_name);

#### Example:

sql

CopyEdit

CREATE INDEX idx\_name

ON students(student\_name);

1. What is the purpose of specifying data types and constraints during table creation?

## Ans. . **Purpose of Data Types**

**Data types** define the kind of data a column can hold (e.g., integers, strings, dates).

### ✅ ****Why Data Types Are Important:****

* 🛡️ **Data Integrity**: Prevents invalid data entry (e.g., text in a numeric field).
* 💾 **Storage Optimization**: Helps the database use memory efficiently.
* ⚡ **Performance**: Improves query speed and processing by optimizing indexing and storage.
* 📊 **Operations Control**: Enables appropriate operations (e.g., arithmetic on INT, date comparisons on DATE).

### 🔸 Example:

sql

CopyEdit

age INT, -- Only whole numbers allowed

student\_name VARCHAR(100), -- String up to 100 characters

admission\_date DATE -- Only date values allowed

## 🔹 2. **Purpose of Constraints**

**Constraints** define rules that restrict the type of data allowed in a column or a table.

### ✅ ****Why Constraints Are Important:****

* 🧮 **Ensure Uniqueness**: Prevent duplicate records (UNIQUE, PRIMARY KEY).
* ❌ **Prevent Nulls**: Ensure required fields are filled (NOT NULL).
* 🔗 **Maintain Relationships**: Link related data between tables (FOREIGN KEY).
* ✅ **Enforce Validity**: Restrict values to logical ranges or sets (CHECK).
* 🧰 **Set Defaults**: Provide default values when none are given (DEFAULT).

### 🔸 Example:

sql

CopyEdit

student\_id INT PRIMARY KEY, -- Uniquely identifies each record

email VARCHAR(100) UNIQUE, -- No duplicate emails

age INT CHECK (age >= 5), -- Only age 5 or older allowed

class VARCHAR(50) NOT NULL -- Class must be specified

1. Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.

Ans. CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

course\_credits INT NOT NULL

);

1. Use the CREATE command to create a database university\_db.

Ans. CREATE DATABASE university\_db;

1. Use the CREATE command to create a database university\_db.

Ans CREATE DATABASE university\_db;

🔍 Explanation:

CREATE DATABASE: SQL command used to create a new database.

university\_db: The name of the database you want to create.

🔄 Next Step (Optional but Recommended)

After creating the database, you should select it before creating tables:

sql

Copy

Edit

USE university\_db;

1. How can you add, modify, and drop columns from a table using ALTER?

## Ans. 1. **Add a Column**

### 🔸 Syntax:

sql

CopyEdit

ALTER TABLE table\_name

ADD column\_name datatype [constraint];

### ✅ Example:

sql

CopyEdit

ALTER TABLE students

ADD email VARCHAR(100) UNIQUE;

Adds an email column to the students table and ensures values are unique.

## 🔹 2. **Modify a Column**

Note: Some databases use MODIFY, others use ALTER COLUMN. Syntax may vary slightly depending on the DBMS (e.g., MySQL, PostgreSQL, SQL Server).

### 🔸 MySQL / Oracle Syntax:

sql

CopyEdit

ALTER TABLE table\_name

MODIFY column\_name new\_datatype [constraint];

### 🔸 SQL Server / PostgreSQL Syntax:

sql

CopyEdit

ALTER TABLE table\_name

ALTER COLUMN column\_name TYPE new\_datatype;

### ✅ Example (MySQL):

sql

CopyEdit

ALTER TABLE students

MODIFY age INT NOT NULL;

## 🔹 3. **Drop a Column**

### 🔸 Syntax:

sql

CopyEdit

ALTER TABLE table\_name

DROP COLUMN column\_name;

### ✅ Example:

sql

CopyEdit

ALTER TABLE students

DROP COLUMN address;

Removes the address column from the students table.

### 📝 Summary Table

| **Action** | **SQL Command Example** |
| --- | --- |
| Add Column | ADD column\_name datatype |
| Modify Column | MODIFY column\_name datatype (MySQL) or ALTER COLUMN column\_name TYPE datatype (PostgreSQL) |
| Drop Column | DROP COLUMN column\_name |

## 1. **Add a Column**

### 🔸 Syntax:

sql

CopyEdit

ALTER TABLE table\_name

ADD column\_name datatype [constraint];

### ✅ Example:

sql

CopyEdit

ALTER TABLE students

ADD email VARCHAR(100) UNIQUE;

Adds an email column to the students table and ensures values are unique.

## 🔹 2. **Modify a Column**

Note: Some databases use MODIFY, others use ALTER COLUMN. Syntax may vary slightly depending on the DBMS (e.g., MySQL, PostgreSQL, SQL Server).

### 🔸 MySQL / Oracle Syntax:

sql

CopyEdit

ALTER TABLE table\_name

MODIFY column\_name new\_datatype [constraint];

### 🔸 SQL Server / PostgreSQL Syntax:

sql

CopyEdit

ALTER TABLE table\_name

ALTER COLUMN column\_name TYPE new\_datatype;

### ✅ Example (MySQL):

sql

CopyEdit

ALTER TABLE students

MODIFY age INT NOT NULL;

## 🔹 3. **Drop a Column**

### 🔸 Syntax:

sql

CopyEdit

ALTER TABLE table\_name

DROP COLUMN column\_name;

### ✅ Example:

sql

CopyEdit

ALTER TABLE students

DROP COLUMN address;

Removes the address column from the students table.

### 📝 Summary Table

| **Action** | **SQL Command Example** |
| --- | --- |
| Add Column | ADD column\_name datatype |
| Modify Column | MODIFY column\_name datatype (MySQL) or ALTER COLUMN column\_name TYPE datatype (PostgreSQL) |
| Drop Column | DROP COLUMN column\_name |

1. Modify the courses table by adding a column course\_duration using the ALTER command.

Ans. ALTER TABLE courses

ADD course\_duration VARCHAR(50);

1. Drop the course\_credits column from the courses table.

Ans. ALTER TABLE courses

DROP COLUMN course\_credits;

1. What is the function of the DROP command in SQL?

Ans. -- Drop a table

DROP TABLE table\_name;

-- Drop a database

DROP DATABASE database\_name;

-- Drop a view

DROP VIEW view\_name;

-- Drop a user-defined function

DROP FUNCTION function\_name;

1. What are the implications of dropping a table from a database?

### Ans. . Permanent Data Loss

* **All data** stored in the table is **permanently deleted**.
* There is **no way to recover** the data unless you have a backup.

**🧱 2. Loss of Table Structure**

* The table's **schema (columns, constraints, indexes, etc.)** is removed.
* You lose the design/structure of the table along with the data.

**🔗 3. Breaks Dependencies**

* Any **views, stored procedures, triggers, or foreign key constraints** that reference the table will become invalid or cause errors.
* You may need to drop or modify dependent objects before dropping the table.

**📉 Impact on Applications**

* Applications or scripts that query the dropped table will fail, leading to **runtime errors**.
* This can result in system downtime or data processing failures.

**🧪 Example**

sql

CopyEdit

DROP TABLE orders;

1. Drop the teachers table from the school\_db database.

Ans DROP TABLE school\_db.teachers;

Important Notes:

This will permanently delete the teachers table and all its data.

Make sure you are connected to the school\_db database or have the necessary privileges.

If you're already using school\_db, you can just run:

sql

Copy

Edit

DROP TABLE teachers;

If you want to avoid an error in case the table doesn’t exist, use:

sql

Copy

Edit

DROP TABLE IF EXISTS teachers;

1. Drop the students table from the school\_db database and verify that the table has been removed

Ans. ROP TABLE IF EXISTS students;

If not, use a fully qualified name (depends on the SQL dialect):

sql

Copy

Edit

DROP TABLE school\_db.students;

✅ Step 2: Verify the Table Has Been Removed

You can verify the removal using one of the following queries (depending on your SQL database system):

For MySQL:

sql

Copy

Edit

SHOW TABLES LIKE 'students';

For PostgreSQL:

sql

Copy

Edit

SELECT tablename FROM pg\_tables WHERE tablename = 'students';

For SQL Server:

sql

Copy

Edit

SELECT \* FROM information\_schema.tables WHERE table\_name = 'students';

For Oracle:

sql

Copy

Edit

SELECT table\_name FROM user\_tables WHERE table\_name = 'STUDENTS';

ROP TABLE IF EXISTS students;

If not, use a fully qualified name (depends on the SQL dialect):

1. Define the INSERT, UPDATE, and DELETE commands in SQL.

Ans. NSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

Example:

sql

Copy

Edit

INSERT INTO students (id, name, age)

VALUES (1, 'Alice', 20);

🔁 UPDATE: Modify Existing Data

Function: Changes data in one or more existing rows.

Syntax:

sql

Copy

Edit

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Example:

sql

Copy

Edit

UPDATE students

SET age = 21

WHERE id = 1;

⚠️ Always use a WHERE clause to avoid updating all rows.

❌ DELETE: Remove Data

Function: Deletes one or more rows from a table.

Syntax:

sql

Copy

Edit

DELETE FROM table\_name

WHERE condition;

Example:

sql

Copy

Edit

DELETE FROM students

WHERE id = 1;

⚠️ Without a WHERE clause, all rows in the table will be deleted:

sql

Copy

Edit

DELETE FROM students; -- Deletes ALL rows!

1. Insert three records into the courses table using the INSERT command.

Ans. INSERT INTO courses (course\_id, course\_name, credits)

VALUES (101, 'Mathematics', 3);

INSERT INTO courses (course\_id, course\_name, credits)

VALUES (102, 'Physics', 4);

INSERT INTO courses (course\_id, course\_name, credits)

VALUES (103, 'Literature', 2);

1. Update the course duration of a specific course using the UPDATE command.

Ans. UPDATE courses

SET duration = new\_value

WHERE course\_id = specific\_value;

1. Delete a course with a specific course\_id from the courses table using the DELETE command.

Ans. DELETE FROM courses

WHERE course\_id = specific\_value;

🧪 Example:

To delete the course with course\_id = 102:

sql

Copy

Edit

DELETE FROM courses

WHERE course\_id = 102;

⚠️ Important:

Always include a WHERE clause to avoid deleting all records.

After deletion, you can verify it with:

sql

Copy

Edit

SELECT \* FROM courses WHERE course\_id = 102;

1. What is the SELECT statement, and how is it used to query data?

Ans. SELECT column1, column2, ...

FROM table\_name

WHERE condition;

1. Explain the use of the ORDER BY and WHERE clauses in SQL queries.

Ans. SELECT column1, column2

FROM table\_name

WHERE condition;

1. Retrieve all courses from the courses table using the SELECT statement.

Ans. SELECT \* FROM courses;

Explanation:

SELECT \* tells SQL to return all columns.

FROM courses specifies the table name you're querying.

If you want to retrieve only specific columns (e.g., course name and course ID), you can specify them like this:

sql

Copy

Edit

SELECT course\_id, course\_name FROM courses;

1. Sort the courses based on course\_duration in descending order using ORDER BY.

Ans SELECT \* FROM courses

ORDER BY course\_duration DESC;

Explanation:

ORDER BY course\_duration tells SQL to sort the results by the course\_duration column.

DESC specifies descending order (from highest to lowest).

If you wanted ascending order instead, you would use ASC (or leave it out, as ascending is the default):

sql

Copy

Edit

SELECT \* FROM courses

ORDER BY course\_duration ASC;

1. : Limit the results of the SELECT query to show only the top two courses using LIMIT.

Ans SELECT \* FROM courses

LIMIT 2;

Example with Sorting:

If you want the top two longest-duration courses, combine ORDER BY with LIMIT:

sql

Copy

Edit

SELECT \* FROM courses

ORDER BY course\_duration DESC

LIMIT 2;

Explanation:

ORDER BY course\_duration DESC sorts courses from longest to shortest.

LIMIT 2 restricts the output to only the first two rows of the sorted results.

1. What is the purpose of GRANT and REVOKE in SQL?

Ans. REVOKE privilege\_type ON object\_name FROM user\_name;

Example:

sql

Copy

Edit

REVOKE INSERT ON courses FROM john;

This removes INSERT permission from john on the courses table, but keeps other permissions (like SELECT) unchanged.

Common Privileges You Can GRANT/REVOKE:

Privilege Description

SELECT Read data from a table or view

INSERT Add new rows

UPDATE Modify existing rows

DELETE Remove rows

ALL All available privileges

1. How do you manage privileges using these commands?

Ans. GRANT privilege1, privilege2 ON object\_name TO user\_or\_role;

Example:

sql

Copy

Edit

GRANT SELECT, INSERT ON students TO alice;

Allows alice to view and insert data in the students table.

✅ 2. Granting Privileges with GRANT OPTION

If you want the user to also be able to pass those privileges on to others, use the WITH GRANT OPTION.

Example:

sql

Copy

Edit

GRANT SELECT ON students TO alice WITH GRANT OPTION;

Alice can now also grant SELECT privilege on students to other users.

🚫 3. Revoking Privileges

Use the REVOKE statement to remove previously granted privileges.

Basic Syntax:

sql

Copy

Edit

REVOKE privilege1, privilege2 ON object\_name FROM user\_or\_role;

Example:

sql

Copy

Edit

REVOKE INSERT ON students FROM alice;

Alice loses the ability to insert data into the students table.

👥 4. Managing Privileges with Roles (Best Practice)

Instead of managing privileges for individual users, assign them to roles, and assign roles to users.

Step 1: Create a Role

sql

Copy

Edit

CREATE ROLE data\_entry;

Step 2: Grant Privileges to Role

sql

Copy

Edit

GRANT INSERT, SELECT ON students TO data\_entry;

Step 3: Assign Role to User

sql

Copy

Edit

GRANT data\_entry TO alice;

Step 4: Revoke Role from User

sql

Copy

Edit

REVOKE data\_entry FROM alice;

1. Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

Ans. CREATE USER user1 IDENTIFIED BY 'password1';

CREATE USER user2 IDENTIFIED BY 'password2';

🔒 Replace 'password1' and 'password2' with secure passwords.

✅ Step 2: Grant SELECT Permission to user1

sql

Copy

Edit

GRANT SELECT ON courses TO user1;

This gives user1 read-only access to the courses table.

❗Note:

In PostgreSQL, the syntax is similar, but if you're using schemas (like public.courses), specify the schema:

sql

Copy

Edit

GRANT SELECT ON TABLE public.courses TO user1;

In MySQL, if the database name is school, use:

sql

Copy

Edit

GRANT SELECT ON school.courses TO 'user1'@'localhost';

🔄 To Check Privileges:

To see what privileges a user has (in MySQL):

sql

Copy

Edit

SHOW GRANTS FOR 'user1'@'localhost';

1. Revoke the INSERT permission from user1 and give it to user2.

Ans. REVOKE INSERT ON courses FROM user1;

This removes the ability for user1 to insert data into the courses table.

✅ 2. Grant INSERT to user2

sql

Copy

Edit

GRANT INSERT ON courses TO user2;

This gives user2 permission to insert data into the courses table.

📌 Example (MySQL Style with Hostname):

If you're using MySQL and your users connect from localhost, use:

sql

Copy

Edit

REVOKE INSERT ON school.courses FROM 'user1'@'localhost';

GRANT INSERT ON school.courses TO 'user2'@'localhost';

1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?

### Ans. ****1. COMMIT****

* **Purpose**: Saves all changes made during the current transaction **permanently** to the database.
* **When to Use**: After a group of operations (INSERT, UPDATE, DELETE) that you want to confirm.

#### **Example**:

sql

CopyEdit

BEGIN; -- or START TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

UPDATE accounts SET balance = balance + 100 WHERE id = 2;

COMMIT;

* Transfers money from one account to another, and commits if both updates succeed.

### ****2. ROLLBACK****

* **Purpose**: Undoes all changes made during the current transaction.
* **When to Use**: If there’s an error or if the user decides to cancel the transaction.

#### **Example**:

sql

CopyEdit

BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

-- Something goes wrong (e.g., constraint violation)

ROLLBACK;

1. Explain how transactions are managed in SQL databases.

And. In SQL, **transactions** are used to execute a series of operations as a **single, atomic unit of work**. Transactions help ensure **data integrity**, even in the face of errors, power loss, or concurrent access.

## 🔄 What Is a Transaction?

A **transaction** is a sequence of one or more SQL statements that are treated as a **single logical unit**. Either **all** operations in the transaction are completed successfully, or **none** of them are applied to the database.

## ⚙️ **Key Transaction Control Commands**

| **Command** | **Description** |
| --- | --- |
| BEGIN or START TRANSACTION | Marks the start of a new transaction |
| COMMIT | Permanently saves all changes made in the transaction |
| ROLLBACK | Undoes all changes made in the transaction |
| SAVEPOINT | Creates a save point to roll back to within a transaction (optional use) |
| ROLLBACK TO SAVEPOINT | Rolls back part of the transaction to a previous state |

## 🔐 **ACID Properties of Transactions**

SQL databases manage transactions using the **ACID** principles:

| **Property** | **Meaning** |
| --- | --- |
| **Atomicity** | All changes in a transaction are completed, or none are applied |
| **Consistency** | Transactions take the database from one valid state to another |
| **Isolation** | Transactions do not interfere with each other |
| **Durability** | Once committed, the changes are permanent—even in case of a crash |

## ✅ **Example of a Transaction**

sql

CopyEdit

START TRANSACTION;

UPDATE accounts SET balance = balance - 500 WHERE account\_id = 1;

UPDATE accounts SET balance = balance + 500 WHERE account\_id = 2;

-- If both updates succeed

COMMIT;

-- If an error occurs

-- ROLLBACK;

1. Insert a few rows into the courses table and use COMMIT to save the changes

And . -- Start the transaction

START TRANSACTION;

-- Insert multiple rows

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (101, 'Mathematics', 6);

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (102, 'Physics', 4);

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (103, 'Computer Science', 8);

-- Save the changes permanently

COMMIT;

1. Insert additional rows, then use ROLLBACK to undo the last insert operation.

Ans. -- Start a new transaction

START TRANSACTION;

-- Insert additional rows

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (104, 'Chemistry', 5);

-- Set a savepoint before the next insert

SAVEPOINT before\_last\_insert;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (105, 'Biology', 6); -- This will be rolled back

-- Roll back the last insert only

ROLLBACK TO SAVEPOINT before\_last\_insert;

-- Commit the first insert (Chemistry) only

COMMIT;

1. Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans. -- Start a new transaction

START TRANSACTION;

-- Insert additional rows

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (104, 'Chemistry', 5);

-- Set a savepoint before the next insert

SAVEPOINT before\_last\_insert;

INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES (105, 'Biology', 6); -- This will be rolled back

-- Roll back the last insert only

ROLLBACK TO SAVEPOINT before\_last\_insert;

-- Commit the first insert (Chemistry) only

COMMIT;

1. Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?

## Ans. **ypes of JOINs in SQL**

| **Join Type** | **Description** |
| --- | --- |
| INNER JOIN | Returns **only matching rows** from both tables. |
| LEFT JOIN | Returns **all rows from the left table** and matching rows from the right. |
| RIGHT JOIN | Returns **all rows from the right table** and matching rows from the left. |
| FULL OUTER JOIN | Returns **all rows from both tables**, matched where possible. |

1. How are joins used to combine data from multiple tables?

Ans. To combine data from multiple tables:

* You identify a **related column** (e.g., student\_id) in both tables.
* You use a JOIN clause in a SELECT statement to pull the data together.

## 🔍 **Example: Joining Two Tables**

### Tables:

#### students:

| **student\_id** | **name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |

#### enrollments:

| **enrollment\_id** | **student\_id** | **course** |
| --- | --- | --- |
| 10 | 1 | Math |
| 11 | 2 | Science |

### ✅ ****SQL Query to Combine Data****

sql

CopyEdit

SELECT students.name, enrollments.course

FROM students

INNER JOIN enrollments ON students.student\_id = enrollments.student\_id;

### 🔄 ****Result:****

| **name** | **course** |
| --- | --- |
| Alice | Math |
| Bob | Science |

## 🧱 **How JOINs Work Under the Hood**

* SQL evaluates the JOIN condition (students.student\_id = enrollments.student\_id) for each combination of rows.
* Only matching pairs are returned (for INNER JOIN).
* Other join types (e.g., LEFT JOIN, RIGHT JOIN) decide **which unmatched rows to keep**.

1. Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

And. -- Insert into departments

INSERT INTO departments (department\_id, department\_name) VALUES

(1, 'Human Resources'),

(2, 'Finance'),

(3, 'Engineering');

-- Insert into employees

INSERT INTO employees (employee\_id, employee\_name, department\_id) VALUES

(101, 'Alice', 1),

(102, 'Bob', 2),

(103, 'Carol', 3),

(104, 'Dave', 2);

1. : Use a LEFT JOIN to show all departments, even those without employees.

Ans. ELECT

departments.department\_name,

employees.employee\_name

FROM

departments

LEFT JOIN

employees ON departments.department\_id = employees.department\_id;

🔍 What This Does:

Lists every department from the departments table.

If a department has employees, their names are shown.

If a department has no employees, employee\_name will be NULL.

🧾 Sample Result:

department\_name employee\_name

Human Resources Alice

Finance Bob

Finance Dave

Engineering Carol

Marketing NULL

Assuming "Marketing" was added to the departments table but has no employees assigned.

➕ (Optional) Add a department with no employees:

sql

Copy

Edit

INSERT INTO departments (department\_id, department\_name)

VALUES (4, 'Marketing');

1. What is the GROUP BY clause in SQL? How is it used with aggregate functions?

Ans. The GROUP BY clause is used to **group rows** that have the same values in specified columns into **summary rows** — like "total sales per region" or "number of students per course". It’s most commonly used with **aggregate functions** such as:

* COUNT() – number of items
* SUM() – total of numeric values
* AVG() – average value
* MIN() / MAX() – smallest/largest value

**✅ Syntax of GROUP BY**

sql

CopyEdit

SELECT column1, AGG\_FUNCTION(column2)

FROM table\_name

GROUP BY column1;

**🧠 Example Scenario**

Assume a table called orders:

| **order\_id** | **customer\_id** | **amount** |
| --- | --- | --- |
| 1 | 101 | 100 |
| 2 | 101 | 150 |
| 3 | 102 | 200 |
| 4 | 103 | 300 |

**🔍 Example: Total amount per customer**

sql

CopyEdit

SELECT customer\_id, SUM(amount) AS total\_spent

FROM orders

GROUP BY customer\_id;

**✅ Result:**

| **customer\_id** | **total\_spent** |
| --- | --- |
| 101 | 250 |
| 102 | 200 |
| 103 | 300 |

1. Explain the difference between GROUP BY and ORDER BY.

### Ans. ****Difference Between**** GROUP BY ****and**** ORDER BY ****in SQL****

GROUP BY and ORDER BY are both used in SQL queries but serve **very different purposes**:

### 🔹 ****1.**** GROUP BY ****– Used for Aggregation****

* **Purpose:** To **group rows** that have the same values in one or more columns.
* **Commonly used with:** Aggregate functions like SUM(), COUNT(), AVG(), MIN(), MAX().
* **Produces:** One result row per group.

#### ✅ Example:

sql

CopyEdit

SELECT department\_id, COUNT(\*) AS num\_employees

FROM employees

GROUP BY department\_id;

🟢 This groups employees by department\_id and counts how many are in each department.

### 🔸 ****2.**** ORDER BY ****– Used for Sorting****

* **Purpose:** To **sort the final result** of a query by one or more columns (or expressions).
* **Commonly used with:** Any query, regardless of grouping or aggregation.
* **Produces:** All rows (or grouped rows), sorted.

#### ✅ Example:

sql

CopyEdit

SELECT \* FROM employees

ORDER BY employee\_name ASC;

🟢 This sorts the full list of employees alphabetically by name.

### 🔄 ****Using**** GROUP BY ****and**** ORDER BY ****Together****

You can group and then sort the grouped results:

sql

CopyEdit

SELECT department\_id, COUNT(\*) AS num\_employees

FROM employees

GROUP BY department\_id

ORDER BY num\_employees DESC;

🟢 This:

1. Groups employees by department,
2. Counts them, and
3. Sorts the result by the number of employees .
4. Group employees by department and count the number of employees in each department using GROUP BY.

Ans. SELECT

department\_id,

COUNT(\*) AS num\_employees

FROM

employees

GROUP BY

department\_id;

🔍 What This Does:

GROUP BY department\_id: groups all rows by each department.

COUNT(\*): counts the number of employees in each department.

AS num\_employees: gives the count column a readable alias.

🧾 Sample Result:

department\_id num\_employees

1 2

2 3

3 1

This assumes your employees table has department data like we used in earlier examples.

➕ Optional: Join to show department names

If you want to display department names as well, use a JOIN:

sql

Copy

Edit

SELECT

d.department\_name,

COUNT(e.employee\_id) AS num\_employees

FROM

departments d

LEFT JOIN

employees e ON d.department\_id = e.department\_id

GROUP BY

d.department\_name;

55 . Use the AVG aggregate function to find the average salary of employees in each department.

Ans. SELECT

department\_id,

AVG(salary) AS average\_salary

FROM

employees

GROUP BY

department\_id;

* 1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?

Ans. A **stored procedure** is a **precompiled set of SQL statements** stored in the database that can be executed repeatedly as a single unit. It is like a function or script saved in the database.

**✅ Key Features of Stored Procedures**

* **Reusable**: Defined once and run many times.
* **Encapsulated logic**: Can contain multiple SQL statements (queries, loops, conditionals).
* **Accepts parameters**: Inputs can be passed to control its behavior.
* **Improves performance**: Compiled once, executed many times.
* **Enhances security**: Access to complex logic without exposing the underlying tables.

**📌 Syntax Example (MySQL style)**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE GetEmployeesByDept(IN dept\_id INT)

BEGIN

SELECT employee\_name, salary

FROM employees

WHERE department\_id = dept\_id;

END //

DELIMITER ;

To call it:

sql

CopyEdit

CALL GetEmployeesByDept(2);

**🆚 Stored Procedure vs. Standard SQL Query**

| **Feature** | **Stored Procedure** | **Standard SQL Query** |
| --- | --- | --- |
| **Definition** | A named, stored block of SQL code | A single, ad-hoc SQL command |
| **Reusability** | Reusable and callable multiple times | Written and executed once |
| **Parameters** | Can accept input/output parameters | No parameters |
| **Complex Logic** | Can contain loops, conditionals | Limited to basic operations |
| **Performance** | Precompiled and optimized | Parsed and compiled each time |
| **Security** | Users can execute without direct access | Direct access |

1. Explain the advantages of using stored procedures.

### Ans. mproved Performance

* **Precompiled:** Stored procedures are compiled once and stored in the database. On subsequent calls, they execute faster than regular SQL queries.
* **Reduced network traffic:** Instead of sending multiple SQL statements from an application, a single call to a stored procedure can execute many operations.

**2. 🔄 Reusability and Maintainability**

* **Centralized logic:** Business rules and SQL logic are encapsulated in one place, making updates easier and consistent across applications.
* **Code reuse:** A stored procedure can be reused in different programs or by different users without rewriting the logic.

**3. 🔒 Enhanced Security**

* **Access control:** Permissions can be granted to execute a procedure without giving direct access to underlying tables.
* **Input validation:** Procedures can validate inputs, reducing the risk of SQL injection and data corruption.

**4. 🧠 Modularity**

* Large or complex operations can be broken down into smaller, manageable stored procedures.
* Encourages separation of concerns: application code handles the UI, while SQL logic stays within the database.

**5. ✅ Consistency**

* Enforces consistent implementation of business rules and calculations.
* Ensures that all users and applications execute the same logic in the same way.

**6. 🔁 Supports Transactions**

* Can include BEGIN, COMMIT, and ROLLBACK to control transactions inside the procedure.
* Ensures **atomic operations** — either all changes succeed or none are applied.

**7. 🛠️ Ease of Maintenance**

* Updates to logic only need to be made in the procedure — not in every application or script that uses it.
* Helps centralize debugging and testing of logic.

**🧾 Summary Table**

| **Advantage** | **Benefit** |
| --- | --- |
| Performance | Faster execution due to precompilation |
| Reusability | Encapsulated logic can be reused |
| Security | Access can be restricted at the procedure level |
| Maintainability | Easier to update logic in one centralized location |
| Modularity | Breaks complex logic into manageable components |
| Consistency | Ensures uniform behavior and results |
| Transaction Support | Manages atomic operations efficiently |

1. Write a stored procedure to retrieve all employees from the employees table based on department.

Ans. DELIMITER //

CREATE PROCEDURE GetEmployeesByDepartment(IN dept\_id INT)

BEGIN

SELECT employee\_id, employee\_name, department\_id, salary

FROM employees

WHERE department\_id = dept\_id;

END //

DELIMITER ;

1. Write a stored procedure that accepts course\_id as input and returns the course details.

Ans. DELIMITER //

CREATE PROCEDURE GetCourseDetails(IN input\_course\_id INT)

BEGIN

SELECT course\_id, course\_name, course\_duration, course\_fee

FROM courses

WHERE course\_id = input\_course\_id;

END //

DELIMITER ;

1. Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans. -- Start a transaction

START TRANSACTION;

-- Insert first employee

INSERT INTO employees (employee\_id, employee\_name, department\_id, salary)

VALUES (201, 'Alice', 1, 60000);

-- Create a savepoint

SAVEPOINT sp1;

-- Insert second employee

INSERT INTO employees (employee\_id, employee\_name, department\_id, salary)

VALUES (202, 'Bob', 2, 55000);

-- Commit changes before the savepoint (Alice's record)

RELEASE SAVEPOINT sp1; -- Optional in some databases

COMMIT;

-- Insert third employee

INSERT INTO employees (employee\_id, employee\_name, department\_id, salary)

VALUES (203, 'Charlie', 3, 58000);

-- Oops! Decide to undo the last insert

ROLLBACK TO SAVEPOINT sp1;

-- Final commit to confirm rollback effect

COMMIT;

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |