# Module 4 – Introduction to DBMS

## Introduction to SQL

### Theory Questions:

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

**ANS.** SQL (Structured Query Language) is a standard language for managing and manipulating relational databases. It's essential because it provides a structured way to interact with databases, allowing users to define, manipulate, control, and query data. It is the standard language across almost all relational database systems.

2. **Explain the difference between DBMS and RDBMS.**

**ANS.** **DBMS (Database Management System):** A software application that interacts with users, other applications, and the database itself to capture and analyze data. Examples include MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. A DBMS may or may not be relational.

**RDBMS (Relational Database Management System):** A type of DBMS that structures data in a tabular format (rows and columns) and establishes relationships between those tables. RDBMS enforces data integrity and consistency using structures like primary keys and foreign keys. All RDBMS are DBMS, but not all DBMS are RDBMS.

3. **Describe the role of SQL in managing relational databases.**

**ANS.** SQL is the primary language used to manage RDBMS. It allows users to:

   - **Create** database structures (tables, views, etc.)
   - **Insert, update, and delete** data within these structures.
   - **Query** the data to retrieve specific information.
   - **Control** access to the data and manage user permissions.
   - **Ensure** data integrity and consistency.

4. **What are the key features of SQL?**

**ANS. Data Definition Language (DDL):** Commands for defining the database schema (CREATE, ALTER, DROP).

   - **Data Manipulation Language (DML):** Commands for manipulating data

(INSERT, UPDATE, DELETE).
- **Data Query Language (DQL):** Commands for querying data (SELECT).
- **Data Control Language (DCL):** Commands for controlling access to data (GRANT, REVOKE).
- **Transaction Control Language (TCL):** Commands for managing transactions (COMMIT, ROLLBACK, SAVEPOINT).
- **Constraints:** Mechanisms to enforce data integrity (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK).
- **Joins:** Ability to combine data from multiple tables.
- **Stored Procedures:** Precompiled SQL code for reusability.
- **Views:** Virtual tables based on the result-set of a SQL statement.
- **Triggers:** Special stored procedures that automatically execute in response to certain events.

**LAB EXERCISES:**

- **Lab 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.**
```
-- Create the database
CREATE DATABASE school_db;

-- Use the database
USE school_db;

-- Create the students table
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(255),
    age INT,
    class VARCHAR(10),
    address VARCHAR(255)
);
```

- **Lab 2: Insert five records into the students table and retrieve all records using the SELECT statement.**
```
-- Insert records into the students table
INSERT INTO students (student_id, student_name, age, class, address) VALUES
(1, 'Alice Smith', 15, '10th', '123 Main St'),
```

```
(2, 'Bob Johnson', 16, '11th', '456 Oak Ave'),
(3, 'Charlie Brown', 14, '9th', '789 Pine Ln'),
(4, 'Diana Miller', 17, '12th', '101 Elm St'),
(5, 'Ethan Davis', 15, '10th', '222 Cedar Rd');

-- Retrieve all records from the students table
SELECT * FROM students;
```

**2. SQL Syntax**

**Theory Questions:**

1. **What are the basic components of SQL syntax?**
2. **ANS.**The basic components of SQL syntax include:
   - **Keywords:** Reserved words with specific meanings (e.g., SELECT, FROM, WHERE).
   - **Identifiers:** Names given to database objects (e.g., table names, column names).
   - **Operators:** Symbols that perform operations (e.g., =, >, <, +, -).
   - **Values:** Specific data, such as numbers or strings (e.g., 10, 'John Doe').
   - **Clauses:** Parts of a SQL statement that perform specific functions (e.g., SELECT clause, FROM clause, WHERE clause).
   - **Statements:** Complete instructions in SQL (e.g., SELECT * FROM students;).
   - **Functions:** Built-in operations, like AVG(), COUNT(), SUM().

3. **Write the general structure of an SQL SELECT statement.**
   **ANS.** The general structure of a SELECT statement is:
   ```
   SELECT column1, column2, ...
   FROM table_name
   WHERE condition
   ORDER BY column1, column2, ...
   LIMIT row_count;
   ```

4. **Explain the role of clauses in SQL statements.**
   **ANS.**Clauses define the actions to be performed in a SQL statement. Common clauses include:
   - SELECT: Specifies the columns to retrieve.

- FROM: Specifies the table to retrieve data from.
- WHERE: Filters records based on a condition.
- ORDER BY: Sorts the result set.
- GROUP BY: Groups rows with the same values in specified columns
- HAVING: Filters grouped records.
- LIMIT: Limits the number of rows returned.

**LAB EXERCISES:**

- **Lab 1: Write SQL queries to retrieve specific columns (student_name and age) from the students table.**
  **ANS.**-- Retrieve student_name and age from the students table
  SELECT student_name, age FROM students;

- **Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.**
  **ANS.**-- Retrieve students whose age is greater than 10
  SELECT * FROM students WHERE age > 10;

**3. SQL Constraints**

**Theory Questions:**

1. **What are constraints in SQL? List and explain the different types of constraints.**
   **ANS.**Constraints are rules enforced on data columns to maintain data integrity and accuracy. Types of constraints include:
   - **PRIMARY KEY:** Uniquely identifies each record in a table; cannot contain NULL values.
   - **FOREIGN KEY:** A field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
   - **UNIQUE:** Ensures that all values in a column are different.
   - **NOT NULL:** Ensures that a column cannot have a NULL value.
   - **CHECK:** Ensures that all values in a column satisfy a specific condition.
   - **DEFAULT:** Sets a default value for a column if no value is specified.
2. **How do PRIMARY KEY and FOREIGN KEY constraints differ?**

**ANS. PRIMARY KEY:** Identifies a unique record *within* a table. A table can have only one primary key.

- **FOREIGN KEY:** Creates a link between two tables. It ensures referential

integrity by requiring that the values in one table match the primary key values in another table.

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

**ANS.NOT NULL:** Ensures that a column always contains a value and cannot be left empty.

- **UNIQUE:** Ensures that all values in a column are distinct. A table can have multiple UNIQUE constraints. A UNIQUE constraint allows one NULL value (in most RDBMS), while a PRIMARY KEY does not allow any NULL values.

**LAB EXERCISES:**

- **Lab 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 the teachers table
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(255) NOT NULL,
    subject VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);
```

- **Lab 2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.**
```
-- Add a foreign key constraint to the students table, referencing teacher_id
ALTER TABLE students
ADD COLUMN teacher_id INT,
ADD FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);

-- Insert some data into teachers
INSERT INTO teachers (teacher_id, teacher_name, subject, email) VALUES
(101, 'Mr. Smith', 'Math', 'smith@example.com'),
(102, 'Ms. Jones', 'English', 'jones@example.com');

-- Update the students table to set the teacher_id for existing students
UPDATE students SET teacher_id = 101 WHERE student_id IN (1, 3);
UPDATE students SET teacher_id = 102 WHERE student_id IN (2, 4, 5);
```

**4. Main SQL Commands and Sub-commands (DDL)**

**Theory Questions:**

1. **Define the SQL Data Definition Language (DDL).**
   **ANS.**DDL (Data Definition Language) is a subset of SQL used to create and modify the structure of database objects, such as tables, schemas, and indexes. DDL commands deal with the *design* of the database.
2. **Explain the CREATE command and its syntax.**
   **ANS.**The CREATE command is used to create new database objects. Its syntax varies depending on the object being created.
   - **Create a database:**
     CREATE DATABASE database_name;

   - **Create a table:**
     CREATE TABLE table_name (
       column1 datatype constraint,
       column2 datatype constraint,
       ...
     );

3. **What is the purpose of specifying data types and constraints during table creation?**
   **ANS.**Specifying data types ensures that the correct type of data is stored in each column (e.g., numbers, strings, dates). Constraints enforce rules on the data to maintain data integrity and consistency. Together, they define the structure and quality of the data within the database.

**LAB EXERCISES:**

- **Lab 1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.**
  **ANS.**-- Create the courses table
  CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(255),
    course_credits INT
  );

- **Lab 2: Use the CREATE command to create a database university_db.**
  **-- Create the university_db database**

CREATE DATABASE university_db;

**5. ALTER Command**

**Theory Questions:**

1. **What is the use of the ALTER command in SQL?**
   **AND.** The ALTER command is used to modify the structure of an existing database object, such as a table. It can be used to add, modify, or delete columns, constraints, and other properties.
2. **How can you add, modify, and drop columns from a table using ALTER?**

**ANS.Add a column:**
ALTER TABLE table_name
ADD column_name datatype constraint;

- **Modify a column (Data Type):** (Note: Modifying data type can be complex and DBMS-specific)
  ALTER TABLE table_name
  MODIFY COLUMN column_name datatype;  -- MySQL
  ALTER TABLE table_name
  ALTER COLUMN column_name datatype;  -- SQL Server

- **Drop a column:**
  ALTER TABLE table_name
  DROP COLUMN column_name;

**LAB EXERCISES:**

- **Lab 1: Modify the courses table by adding a column course_duration using the ALTER command.**
  -- Add the course_duration column to the courses table
  ALTER TABLE courses
  ADD course_duration INT;

- **Lab 2: Drop the course_credits column from the courses table.**
  -- Drop the course_credits column from the courses table
  ALTER TABLE courses
  DROP COLUMN course_credits;

**6. DROP Command**

**Theory Questions:**

1. **What is the function of the DROP command in SQL?**
   **ANS.**The DROP command is used to completely remove a database object, such as a table, view, or database.

2. **What are the implications of dropping a table from a database?**
   **ANS.**Dropping a table results in the permanent deletion of the table and all its data. Any relationships (e.g., foreign key constraints) that depend on the dropped table may also be affected, potentially causing errors or data loss in other tables. This action cannot be undone with a simple SQL command (though some RDBMS have recovery mechanisms).

**LAB EXERCISES:**

- **Lab 1:** Drop the teachers table from the school_db database.
  -- Drop the teachers table
  DROP TABLE teachers;

- **Lab 2:** Drop the students table from the school_db database and verify that the table has been removed.
  -- Drop the students table
  DROP TABLE students;

  -- Attempt to select from the table (this will cause an error if the table is dropped)
  SELECT * FROM students;

**7. Data Manipulation Language (DML)**

**Theory Questions:**

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

**ANS. INSERT:** Adds new records to a table.

- **UPDATE:** Modifies existing records in a table.
- **DELETE:** Removes records from a table.

2. **What is the importance of the WHERE clause in UPDATE and DELETE operations?**
   **ANS.** The WHERE clause is crucial in UPDATE and DELETE operations because it specifies which records should be modified or removed. If the WHERE clause is omitted, UPDATE will modify *all* rows in the table, and DELETE will remove *all* rows, which is rarely what is intended.

**LAB EXERCISES:**

- **Lab 1: Insert three records into the courses table using the INSERT command.**
  ```
  -- Insert records into the courses table
  INSERT INTO courses (course_id, course_name, course_duration) VALUES
  (101, 'Introduction to SQL', 3),
  (102, 'Database Design', 4),
  (103, 'Advanced SQL', 3);
  ```

- **Lab 2: Update the course duration of a specific course using the UPDATE command.**
  ```
  -- Update the course duration for course_id 102
  UPDATE courses
  SET course_duration = 5
  WHERE course_id = 102;
  ```

- **Lab 3: Delete a course with a specific course_id from the courses table using the DELETE command.**
  ```
  -- Delete the course with course_id 103
  DELETE FROM courses
  WHERE course_id = 103;
  ```

**8. Data Query Language (DQL)**

**Theory Questions:**

1. **What is the SELECT statement, and how is it used to query data?**
   **ANS.**The SELECT statement is the fundamental DQL command used to retrieve data from one or more tables. It specifies which columns to retrieve and can filter, sort, and group the results.
2. **Explain the use of the ORDER BY and WHERE clauses in SQL queries.**

**ANS.ORDER BY**: Sorts the result set of a query in ascending (ASC) or descending (DESC) order based on the values in one or more columns.

- **WHERE**: Filters the records returned by a query, based on specified conditions.

**LAB EXERCISES:**

- **Lab 1: Retrieve all courses from the courses table using the SELECT statement.**

```
-- Retrieve all courses
SELECT * FROM courses;
```

- **Lab 2: Sort the courses based on course_duration in descending order using ORDER BY.**

```
-- Sort courses by course_duration in descending order
SELECT * FROM courses
ORDER BY course_duration DESC;
```

- **Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.**

```
-- Limit the results to the top 2 courses
SELECT * FROM courses
LIMIT 2;
```

**9. Data Control Language (DCL)**

**Theory Questions:**

1. **What is the purpose of GRANT and REVOKE in SQL?**
   **ANS.** GRANT and REVOKE are DCL commands used to manage user privileges and permissions on database objects.
   - GRANT: Assigns specific privileges to a user.
   - REVOKE: Removes privileges from a user.
2. **How do you manage privileges using these commands?**
   **ANS.** Privileges are managed by specifying the actions a user can perform (e.g., SELECT, INSERT, UPDATE, DELETE) on specific database objects (e.g., tables, views). GRANT assigns these privileges, and REVOKE takes them away.

**LAB EXERCISES:**

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

```
-- Create user1;  --  Syntax varies
-- Create user2;  --  Syntax varies

-- Grant SELECT permission to user1 on the courses table
GRANT SELECT ON courses TO user1;
```

- **Lab 2: Revoke the INSERT permission from user1 and give it to user2.**

```
-- Revoke INSERT permission from user1
REVOKE INSERT ON courses FROM user1;

-- Grant INSERT permission to user2
GRANT INSERT ON courses TO user2;
```

**10. Transaction Control Language (TCL)**

**Theory Questions:**

1. **What is the purpose of the COMMIT and ROLLBACK commands in SQL?**
   **ANS.** COMMIT and ROLLBACK are TCL commands used to manage transactions:
   - COMMIT: Saves all changes made within a transaction permanently.
   - ROLLBACK: Undoes all changes made within a transaction, reverting the database to its state before the transaction began.
2. **Explain how transactions are managed in SQL databases.**
   **ANS.** A transaction is a sequence of SQL operations that are treated as a single unit. Transactions ensure data consistency by adhering to ACID properties (Atomicity, Consistency, Isolation, Durability). COMMIT and ROLLBACK control whether the changes within a transaction are saved or discarded.

**LAB EXERCISES:**

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

```
-- Start a transaction (Implicit in many systems, explicit in some)
--  BEGIN TRANSACTION;  --  For some systems

-- Insert rows
INSERT INTO courses (course_id, course_name, course_duration) VALUES
(201, 'Web Development', 4),
(202, 'Data Science', 5);

-- Commit the transaction
```

COMMIT;

- **Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.**
  -- Start a transaction (Implicit)
  --  BEGIN TRANSACTION;

  -- Insert rows
  INSERT INTO courses (course_id, course_name, course_duration) VALUES
  (203, 'Machine Learning', 6),
  (204, 'Cloud Computing', 4);

  -- Rollback the transaction
  ROLLBACK;

- **Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**
  -- Start a transaction
  -- BEGIN TRANSACTION;

  -- Create a savepoint
  SAVEPOINT before_update;

  -- Update rows
  UPDATE courses
  SET course_duration = course_duration + 1;

  -- Rollback to the savepoint
  ROLLBACK TO before_update;

  -- Commit (or further changes)
  -- COMMIT;

**11. SQL Joins**

**Theory Questions:**

1. **Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?**
   **ANS.** A JOIN is used to combine rows from two or more tables based on a related

column between them.
    - **INNER JOIN**: Returns only the rows that have matching values in both tables.
    - **LEFT JOIN** (**LEFT OUTER JOIN**): Returns all rows from the left table and the matching rows from the right table. If there's no match in the right table, NULL values are returned for the right table's columns.
    - **RIGHT JOIN** (**RIGHT OUTER JOIN**): Returns all rows from the right table and the matching rows from the left table. If there's no match in the left table, NULL values are returned for the left table's columns.
    - **FULL OUTER JOIN**: Returns all rows from both tables, with NULL values for columns where there is no match.
2. **How are joins used to combine data from multiple tables?**
   **ANS.** Joins combine data from multiple tables by using a common column (usually a primary key in one table and a foreign key in another) to establish a relationship between the tables. This allows you to retrieve related data from different tables in a single query.

**LAB EXERCISES:**

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

```
-- Create the departments table
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(255)
);

-- Create the employees table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(255),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);

-- Insert data into departments
INSERT INTO departments (dept_id, dept_name) VALUES
(1, 'Sales'),
(2, 'Marketing'),
(3, 'Engineering');
```

```
-- Insert data into employees
INSERT INTO employees (emp_id, emp_name, dept_id) VALUES
(101, 'Alice', 1),
(102, 'Bob', 1),
(103, 'Charlie', 2),
(104, 'David', 3),
(105, 'Eve', 3);

-- Perform an INNER JOIN
SELECT employees.emp_name, departments.dept_name
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id;
```

- **Lab 2: Use a LEFT JOIN to show all departments, even those without employees.**

```
-- Perform a LEFT JOIN
SELECT departments.dept_name, employees.emp_name
FROM departments
LEFT JOIN employees ON departments.dept_id = employees.dept_id;
```

**12. SQL Group By**

**Theory Questions:**

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 "find the number of customers in each country". It is often used with aggregate functions (e.g., COUNT, SUM, AVG, MIN, MAX) to calculate summary statistics for each group.
2. **Explain the difference between GROUP BY and ORDER BY.**

**ANS. GROUP BY**: Groups rows with the same values in specified columns *before* aggregate functions are applied.

- **ORDER BY**: Sorts the result set of a query *after* the data has been retrieved and any aggregation has been performed.

**LAB EXERCISES:**

- **Lab 1: Group employees by department and count the number of employees in each department using GROUP BY.**

```
-- Group employees by department and count them
SELECT departments.dept_name, COUNT(employees.emp_id) AS num_employees
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id
GROUP BY departments.dept_name;
```

- **Lab 2: Use the AVG aggregate function to find the average salary of employees in each department. (First, add a salary column to the employees table.)**

```
-- Add a salary column to the employees table
ALTER TABLE employees
ADD salary DECIMAL(10, 2);

-- Update the employees table with salary data
UPDATE employees SET salary = 50000 WHERE emp_id = 101;
UPDATE employees SET salary = 60000 WHERE emp_id = 102;
UPDATE employees SET salary = 55000 WHERE emp_id = 103;
UPDATE employees SET salary = 70000 WHERE emp_id = 104;
UPDATE employees SET salary = 65000 WHERE emp_id = 105;

-- Find the average salary for each department
SELECT departments.dept_name, AVG(employees.salary) AS average_salary
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id
GROUP BY departments.dept_name;
```

**13. SQL Stored Procedure**

**Theory Questions:**

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. Unlike a standard SQL query, which is executed directly each time, a stored procedure is stored and can be executed multiple times by calling its name.
2. **Explain the advantages of using stored procedures.**

**ANS. Improved performance:** Stored procedures are precompiled, reducing execution time.

- **Reduced network traffic:** The entire procedure is sent to the database server once and then executed on the server, rather than sending multiple individual queries.
- **Code reusability:** Stored procedures can be called from multiple applications.
- **Enhanced security:** Permissions can be granted to execute a stored procedure without granting access to the underlying tables.
- **Data integrity:** Stored procedures can enforce business rules and data validation.

**LAB EXERCISES:**

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

```
-- MySQL Example
DELIMITER //
CREATE PROCEDURE GetEmployeesByDepartment (IN dept_name
VARCHAR(255))
BEGIN
   SELECT employees.emp_name
   FROM employees
   INNER JOIN departments ON employees.dept_id = departments.dept_id
   WHERE departments.dept_name = dept_name;
END //
DELIMITER ;

-- Call the stored procedure
CALL GetEmployeesByDepartment('Sales');
```

- **Lab 2: Write a stored procedure that accepts course_id as input and returns the course details.**

```
-- MySQL Example
DELIMITER //
CREATE PROCEDURE GetCourseDetails (IN course_id INT)
BEGIN
   SELECT course_name, course_duration
   FROM courses
   WHERE course_id = course_id;
END //
DELIMITER ;
```

```
-- Call the stored procedure
CALL GetCourseDetails(101);
```

**14. SQL View**

**Theory Questions:**

1. **What is a view in SQL, and how is it different from a table?**
   **ANS.** A view is a virtual table based on the result-set of a SQL statement. Unlike a table, a view does not store data physically. It's essentially a stored query. When you query a view, the underlying SQL statement is executed, and the result is returned.
2. **Explain the advantages of using views in SQL databases.**

**ANS. Data security:** Views can restrict access to certain rows or columns of a table.

- **Simplified queries:** Views can hide complex queries, making it easier for users to retrieve data.
- **Data independence:** Views can insulate applications from changes in the underlying table structure.
- **Data consistency:** Views can ensure that users see a consistent representation of the data.

**LAB EXERCISES:**

- **Lab 1: Create a view to show all employees along with their department names.**

```
-- Create a view
CREATE VIEW employee_department_view AS
SELECT employees.emp_name, departments.dept_name
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id;

-- Query the view
SELECT * FROM employee_department_view;
```

- **Lab 2: Modify the view to exclude employees whose salaries are below $50,000.**

```
--  The original view does not have salary.  We need to recreate it.
-- Recreate the view with salary information and filter.
CREATE OR REPLACE VIEW employee_department_view AS
```

```
SELECT employees.emp_name, departments.dept_name, employees.salary
FROM employees
INNER JOIN departments ON employees.dept_id = departments.dept_id
WHERE employees.salary >= 50000;

-- Query the modified view
SELECT * FROM employee_department_view;
```

**15. SQL Triggers**

**Theory Questions:**

1. **What is a trigger in SQL? Describe its types and when they are used.**
   **ANS.** A trigger is a special type of stored procedure that automatically executes in response to a specific event on a table.
   Types of triggers (based on timing):
   - **BEFORE trigger:** Executes *before* the triggering event.
   - **AFTER trigger:** Executes *after* the triggering event.

   Types of triggers (based on the event):
   - **INSERT trigger:** Activated when a new row is inserted into a table.
   - **UPDATE trigger:** Activated when a row in a table is updated.
   - **DELETE trigger:** Activated when a row is deleted from a table.

   Triggers are used for:
   - Auditing changes to data
   - Enforcing complex business rules
   - Maintaining data integrity
   - Automating tasks
2. **Explain the difference between INSERT, UPDATE, and DELETE triggers.**
   **ANS.** The difference lies in the event that causes the trigger to execute:
   - INSERT trigger: Fires when a new row is added.
   - UPDATE trigger: Fires when an existing row is modified.
   - DELETE trigger: Fires when a row is removed.

**LAB EXERCISES:**

- **Lab 1: Create a trigger to automatically log changes to the employees table when a new employee is added.**

```
-- MySQL Example
CREATE TABLE employee_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    emp_id INT,
    event_type VARCHAR(20),
    event_date TIMESTAMP
);

DELIMITER //
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, event_type, event_date)
    VALUES (NEW.emp_id, 'INSERT', NOW());
END //
DELIMITER ;

-- Insert a new employee to activate the trigger
INSERT INTO employees (emp_id, emp_name, dept_id, salary) VALUES
(106, 'Fiona', 2, 58000);

-- Check the employee_log table
SELECT * FROM employee_log;
```

- **Lab 2: Create a trigger to update the last_modified timestamp whenever an employee record is updated.**

```
-- Add a last_modified column to the employees table
ALTER TABLE employees
ADD last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP;

-- Create the update trigger
-- MySQL Example
DELIMITER //
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    SET NEW.last_modified = NOW();
END //
DELIMITER ;
```

```
-- Update an employee record
UPDATE employees SET salary = 62000 WHERE emp_id = 101;

-- Check the employees table to see the updated last_modified value
SELECT * FROM employees WHERE emp_id = 101;
```

**16. Introduction to PL/SQL**

**Theory Questions:**

1. **What is PL/SQL, and how does it extend SQL's capabilities?**
   **ANS.** PL/SQL (Procedural Language/SQL) is Oracle's extension to standard SQL. It combines the data manipulation capabilities of SQL with procedural programming constructs, such as variables, control structures (loops, conditions), and error handling. PL/SQL allows you to write more complex and sophisticated database applications.
2. **List and explain the benefits of using PL/SQL.**

**ANS. Procedural capabilities:** PL/SQL allows you to write code with control flow, variables, and logic, which is not possible with standard SQL.

- **Improved performance:** PL/SQL code can be stored and executed on the database server, reducing network traffic.
- **Modularity:** PL/SQL code can be organized into blocks, procedures, functions, and packages for better code management and reusability.
- **Error handling:** PL/SQL provides mechanisms for handling exceptions (errors) gracefully.
- **Security:** PL/SQL code can be stored in the database, and permissions can be granted to execute it without granting access to the underlying tables.
- **Integration with SQL:** PL/SQL seamlessly integrates with SQL, allowing you to execute SQL statements within your PL/SQL code.

**LAB EXERCISES:**

- **Lab 1: Write a PL/SQL block to print the total number of employees from the employees table.**

```
-- PL/SQL block to print the total number of employees
DECLARE
   total_employees NUMBER;
BEGIN
   SELECT COUNT(*) INTO total_employees FROM employees;
```

```
    DBMS_OUTPUT.PUT_LINE('Total number of employees: ' || total_employees);
END;
/
```

- **Lab 2: Create a PL/SQL block that calculates the total sales from an orders table. (Assume you have an orders table with columns like order_id, amount, etc.)**

```
-- PL/SQL block to calculate total sales
DECLARE
    total_sales NUMBER;
BEGIN
    SELECT SUM(amount) INTO total_sales FROM orders;
    DBMS_OUTPUT.PUT_LINE('Total sales: ' || total_sales);
END;
/
```

**17. PL/SQL Control Structures**

**Theory Questions:**

1. **What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.**
   **ANS.** Control structures in PL/SQL allow you to control the flow of execution of your code.
   - **IF-THEN**: Executes a block of code based on a condition. Variations include IF-THEN-ELSE and IF-ELSIF-ELSE.
   - **LOOP**: Executes a block of code repeatedly. Types include FOR LOOP, WHILE LOOP, and BASIC LOOP.
2. **How do control structures in PL/SQL help in writing complex queries?**
   **ANS.** Control structures enable you to:
   - Implement conditional logic (e.g., execute different SQL statements based on data values).
   - Iterate over data (e.g., process each row in a result set).
   - Handle different scenarios and exceptions.
   - Break down complex tasks into smaller, manageable blocks of code.

**LAB EXERCISES:**

- **Lab 1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.**

```
-- PL/SQL block to check employee department
```

```
DECLARE
   employee_name VARCHAR2(255);
   employee_dept VARCHAR2(255);
BEGIN
   SELECT emp_name, dept_name INTO employee_name, employee_dept
   FROM employees e JOIN departments d ON e.dept_id = d.dept_id
   WHERE emp_id = 101;  -- Change emp_id to test different employees

   DBMS_OUTPUT.PUT_LINE('Employee Name: ' || employee_name);
   IF employee_dept = 'Sales' THEN
      DBMS_OUTPUT.PUT_LINE('Employee is in the Sales department.');
   ELSIF employee_dept = 'Marketing' THEN
      DBMS_OUTPUT.PUT_LINE('Employee is in the Marketing department.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Employee is in another department.');
   END IF;
END;
/
```

- **Lab 2: Use a FOR LOOP to iterate through employee records and display their names.**

```
-- PL/SQL block to display employee names using a FOR LOOP
BEGIN
   FOR emp_record IN (SELECT emp_name FROM employees) LOOP
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.emp_name);
   END LOOP;
END;
/
```

**18. SQL Cursors**

**Theory Questions:**

1. **What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.**
   **ANS.** A cursor is a pointer to a memory area called the context area, which contains information about a SQL statement and its processing. It allows you to process the rows returned by a query one at a time.
   - **Implicit cursor:** Created and managed automatically by Oracle for SQL statements that return a single row (e.g., SELECT INTO). You don't declare or

open them.
    - **Explicit cursor:** Declared and managed by the programmer for SQL statements that return multiple rows. You need to explicitly declare, open, fetch from, and close them.
2. **When would you use an explicit cursor over an implicit one?**
   **ANS.** You use an explicit cursor when you need to process multiple rows returned by a query. Implicit cursors are suitable for single-row queries, but explicit cursors provide the control necessary to iterate through and manipulate a set of rows.

**LAB EXERCISES:**

- **Lab 1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.**

```
-- PL/SQL block to display employee details using an explicit cursor
DECLARE
  CURSOR emp_cursor IS
    SELECT emp_name, dept_name, salary
    FROM employees e JOIN departments d ON e.dept_id = d.dept_id;
  emp_name_var VARCHAR2(255);
  dept_name_var VARCHAR2(255);
  salary_var NUMBER;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_name_var, dept_name_var, salary_var;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name_var || ', Department: ' ||
dept_name_var || ', Salary: ' || salary_var);
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

- **Lab 2: Create a cursor to retrieve all courses and display them one by one.**

```
-- PL/SQL block to display courses using an explicit cursor
DECLARE
  CURSOR course_cursor IS
    SELECT course_name, course_duration
    FROM courses;
  course_name_var VARCHAR2(255);
```

```
    course_duration_var NUMBER;
BEGIN
    OPEN course_cursor;
    LOOP
        FETCH course_cursor INTO course_name_var, course_duration_var;
        EXIT WHEN course_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Course: ' || course_name_var || ', Duration: ' ||
course_duration_var);
    END LOOP;
    CLOSE course_cursor;
END;
/
```

**19. Rollback and Commit Savepoint**

**Theory Questions:**

1. **Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?**
   **ANS.** A SAVEPOINT is a marker within a transaction that allows you to roll back to a specific point in the transaction, rather than rolling back the entire transaction.
   - SAVEPOINT: Defines a point to which a transaction can be rolled back.
   - ROLLBACK TO savepoint_name: Rolls back the transaction to the specified savepoint, undoing changes made *after* the savepoint.
   - ROLLBACK: Rolls back the entire transaction.
   - COMMIT: Commits the entire transaction, including all changes made before and after any savepoints.
2. **When is it useful to use savepoints in a database transaction?**
   **ANS.** Savepoints are useful when:
   - You have a long transaction with multiple steps, and you want to be able to undo some steps without undoing everything.
   - You want to implement complex error handling or branching logic within a transaction.
   - You want to perform a series of operations, and if one fails, you want to undo only the failed operation and subsequent operations, not the entire transaction.

**LAB EXERCISES:**

- **Lab 1: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.**

```
-- PL/SQL block to demonstrate SAVEPOINT and ROLLBACK TO
BEGIN
   -- Start a transaction (implicit in PL/SQL)

   SAVEPOINT before_insert;

   INSERT INTO courses (course_id, course_name, course_duration) VALUES
   (301, 'Database Administration', 4);
   INSERT INTO courses (course_id, course_name, course_duration) VALUES
   (302, 'Network Security', 3);

   ROLLBACK TO before_insert;  -- Undo the inserts

   -- The inserts are undone, but the transaction is still active.
   -- You could now insert different data or commit.
   INSERT INTO courses (course_id, course_name, course_duration) VALUES
   (301, 'Database Administration', 4);
   COMMIT;
END;
/
```

- **Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.**

```
-- PL/SQL block to demonstrate SAVEPOINT, COMMIT, and ROLLBACK
BEGIN
   -- Start a transaction

   SAVEPOINT before_update;

   UPDATE courses SET course_duration = 5 WHERE course_id = 101;
   COMMIT;  -- Commit the update

   --  The update is now permanent.

   UPDATE courses SET course_duration = 6 WHERE course_id = 102;
   ROLLBACK; -- Rollback only the second update

   -- The first update (course_id = 101) remains committed.
END;
```