Module 4 – Introduction to DBMS

Introduction to SQL

Theory Questions:

1. What is SQL, and why is it essential in database management?
    - SQL serves as the standard language for interacting with relational database management systems. Its importance lies in providing a uniform method to define, manipulate, control, and query data, making it indispensable for nearly all relational database systems.
2. Explain the difference between DBMS and RDBMS.
    - A Database Management System (DBMS) is a software application designed to manage data, allowing for its capture and analysis. Examples include MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. A Relational Database Management System (RDBMS) is a specific type of DBMS that organizes data into tables with defined relationships, ensuring data integrity through mechanisms like primary and foreign keys. Essentially, an RDBMS is a specialized form of a DBMS.
3. Describe the role of SQL in managing relational databases.
    - SQL plays a central role in RDBMS management by enabling users to structure databases, modify data within those structures, retrieve specific information through queries, regulate data access, and maintain data consistency.
4. What are the key features of SQL?
    - SQL is characterized by several key features:
        - DDL (Data Definition Language): For defining the database schema.
        - DML (Data Manipulation Language): For handling data manipulation.
        - DQL (Data Query Language): For querying data.
        - DCL (Data Control Language): For managing data access.
        - TCL (Transaction Control Language): For overseeing transactions.
        - Constraints: To enforce data integrity.
        - Joins: To combine data across tables.
        - Stored Procedures: For reusable code.
        - Views: To create virtual tables.
        - Triggers: To execute code in response to 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.

```
CREATE DATABASE school_db;

USE school_db;

CREATE TABLE students (
   student_id INT,
   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 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', '202 Cedar Rd');

SELECT * FROM students;
```

SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?
   - SQL syntax comprises several fundamental elements:
     - Keywords: Reserved terms with predefined meanings.
     - Identifiers: Names assigned to database objects.
     - Operators: Symbols that perform operations.
     - Values: Specific data entities.
     - Clauses: Components of SQL statements.
     - Statements: Complete SQL instructions.
     - Functions: Built-in operations.
2. Write the general structure of an SQL SELECT statement.
   - The basic structure of a SELECT statement is:

```
SELECT column1, column2, ...
```

```
FROM table_name
WHERE condition
ORDER BY column1, column2, ...
LIMIT row_count;
```

- Note that not all clauses are mandatory.

3. Explain the role of clauses in SQL statements.
    - Clauses are essential components within SQL statements that specify the actions to be performed. Common clauses include SELECT (to specify columns), FROM (to indicate the table), WHERE (to filter records), ORDER BY (to sort results), GROUP BY (to group rows), HAVING (to filter groups), and LIMIT (to restrict row numbers).

LAB EXERCISES:

- Lab 1: Write SQL queries to retrieve specific columns (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.
  SELECT * FROM students WHERE age > 10;

SQL Constraints

Theory Questions:

1. What are constraints in SQL? List and explain the different types of constraints.
    - SQL constraints are rules applied to data columns to ensure data accuracy and integrity. They enforce adherence to specific rules, maintaining database reliability. Constraint types include:
        - PRIMARY KEY: For unique record identification within a table.
        - FOREIGN KEY: To link tables and enforce referential integrity.
        - UNIQUE: To ensure distinct values within a column.
        - NOT NULL: To prevent null values in a column.
        - CHECK: To ensure values meet a specified condition.
        - DEFAULT: To assign a default value when none is provided.
2. How do PRIMARY KEY and FOREIGN KEY constraints differ?
    - A PRIMARY KEY uniquely identifies a record in its own table, and a table can only have one. A FOREIGN KEY, conversely, establishes a link between tables, ensuring that values in one table correspond to primary key values in another, maintaining referential integrity.

3. What is the role of NOT NULL and UNIQUE constraints?
   - A NOT NULL constraint ensures a column always has a value, while a UNIQUE constraint ensures that all values in a column are distinct. Notably, a UNIQUE constraint typically allows one null value, unlike a PRIMARY KEY.

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).

```
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.

```
ALTER TABLE students
ADD COLUMN teacher_id INT,
ADD 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).
   - DDL is a subset of SQL focused on creating and modifying the structure of database objects. DDL commands handle the design and schema of the database.
2. Explain the CREATE command and its syntax.
   - The CREATE command is used to make new database objects, and its syntax varies. For a database: CREATE DATABASE database_name; For 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?
   - Data types ensure that each column stores the correct kind of data, while constraints enforce rules to maintain data integrity and consistency, collectively defining the structure and quality of the data.

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.
```
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 DATABASE university_db;
```

ALTER Command

Theory Questions:

1. What is the use of the ALTER command in SQL?
    - The ALTER command is employed to modify the structure of existing database objects. It allows for adding, modifying, or deleting columns, constraints, and other object properties.
2. How can you add, modify, and drop columns from a table using ALTER?
    - To add a column:
    ```
    ALTER TABLE table_name
    ADD column_name datatype constraint;
    ```
    - To modify a column's data type:
    ```
    ALTER TABLE table_name
    MODIFY COLUMN column_name datatype;  -- MySQL
    ALTER TABLE table_name
    ALTER COLUMN column_name datatype;  -- SQL Server
    ```
    - To 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.
```
ALTER TABLE courses
```

```
ADD COLUMN course_duration VARCHAR(50);
```

- Lab 2: Drop the course_credits column from the courses table.

```
ALTER TABLE courses
DROP COLUMN course_credits;
```

DROP Command

Theory Questions:

1. What is the function of the DROP command in SQL?
    - The DROP command removes a database object, such as a table, view, or database, permanently from the system.
2. What are the implications of dropping a table from a database?
    - Dropping a table permanently deletes it and its data. This can affect relationships with other tables and may lead to errors or data loss in those tables. The action is irreversible through standard SQL.

LAB EXERCISES:

- Lab 1: Drop the teachers table from the school_db database.

```
USE school_db;
DROP TABLE teachers;
```

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

```
USE school_db;
DROP TABLE students;

--  To verify, you can try to select from the table.  This will give an error if it is dropped.
SELECT * FROM students;
```

Data Manipulation Language (DML)

Theory Questions:

1. Define the INSERT, UPDATE, and DELETE commands in SQL.
    - INSERT: Adds new records to a table.
    - UPDATE: Modifies existing table records.
    - DELETE: Removes records from a table.

2. What is the importance of the WHERE clause in UPDATE and DELETE operations?
   - The WHERE clause is vital in UPDATE and DELETE operations as it specifies which records to modify or remove. Omitting it results in all table rows being affected, which is seldom the desired outcome.

LAB EXERCISES:

- Lab 1: Insert three records into the courses table using the INSERT command.
  ```
  INSERT INTO courses (course_id, course_name, course_duration) VALUES
  (101, 'Introduction to SQL', '3 weeks'),
  (102, 'Database Management', '6 weeks'),
  (103, 'Advanced SQL', '4 weeks');
  ```

- Lab 2: Update the course duration of a specific course using the UPDATE command.
  ```
  UPDATE courses
  SET course_duration = '8 weeks'
  WHERE course_id = 102;
  ```

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

Data Query Language (DQL)

Theory Questions:

1. What is the SELECT statement, and how is it used to query data?
   - The SELECT statement is the core DQL command for retrieving data. It specifies the columns to retrieve and can filter, sort, and group results.
2. Explain the use of the ORDER BY and WHERE clauses in SQL queries.
   - ORDER BY: Arranges a query's result set in ascending or descending order based on column values.
   - WHERE: Filters the records that a query returns, based on specified criteria.

LAB EXERCISES:

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

- Lab 2: Sort the courses based on course_duration in descending order using

ORDER BY.
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.
SELECT * FROM courses
LIMIT 2;

Data Control Language (DCL)

Theory Questions:

1. What is the purpose of GRANT and REVOKE in SQL?
   - GRANT and REVOKE are DCL commands for managing user privileges on database objects. GRANT assigns privileges, and REVOKE removes them.
2. How do you manage privileges using these commands?
   - Privileges are managed by specifying the actions a user can perform on database objects. GRANT assigns these, and REVOKE removes them.

LAB EXERCISES:

- Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table. (Note: The exact syntax for creating users varies significantly between database systems (MySQL, PostgreSQL, SQL Server, etc.). This is a general example and you'll need to adapt it.)
-- This is a very general example. Adapt to your specific database.
CREATE USER user1 WITH PASSWORD 'password1';
CREATE USER user2 WITH PASSWORD 'password2';

  GRANT SELECT ON courses TO user1;

- Lab 2: Revoke the INSERT permission from user1 and give it to user2. (Again, adapt to your system)
REVOKE INSERT ON courses FROM user1;
GRANT INSERT ON courses TO user2;

Transaction Control Language (TCL)

Theory Questions:

1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?
   - COMMIT and ROLLBACK, TCL commands, manage transactions, which are sequences of operations treated as a single unit. COMMIT saves transaction changes, and ROLLBACK undoes them.
2. Explain how transactions are managed in SQL databases.
   - SQL databases manage transactions as sequences of operations that adhere to ACID properties (Atomicity, Consistency, Isolation, Durability). COMMIT and ROLLBACK dictate whether transaction changes are saved or discarded.

LAB EXERCISES:

- Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.
  INSERT INTO courses (course_id, course_name, course_duration) VALUES (201, 'Database Design', '4 weeks');
  INSERT INTO courses (course_id, course_name, course_duration) VALUES (202, 'SQL Fundamentals', '2 weeks');
  COMMIT;

- Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.
  INSERT INTO courses (course_id, course_name, course_duration) VALUES (203, 'NoSQL Basics', '1 week');
  ROLLBACK;

- Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.
  SAVEPOINT update_start;
  UPDATE courses SET course_duration = '5 weeks' WHERE course_id = 201;
  ROLLBACK TO update_start; -- This will undo the duration change.
  COMMIT; -- This will save any changes that were not rolled back.

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?
   - A JOIN combines rows from multiple tables based on related columns.
     - INNER JOIN: Returns matching rows from both tables.
     - LEFT JOIN: Returns all left table rows and matching right table rows.

        - RIGHT JOIN: Returns all right table rows and matching left table rows.
        - FULL OUTER JOIN: Returns all rows from both tables.
2. How are joins used to combine data from multiple tables?
    - Joins use common columns, often primary and foreign keys, to relate tables and retrieve combined data 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 TABLE departments (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(255)
);

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

INSERT INTO departments (dept_id, dept_name) VALUES
(1, 'Sales'),
(2, 'Marketing'),
(3, 'Engineering');

INSERT INTO employees (emp_id, emp_name, dept_id) VALUES
(101, 'John Doe', 1),
(102, 'Jane Smith', 2),
(103, 'Mike Johnson', 1),
(104, 'Alice Brown', 3);

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.

```
SELECT departments.dept_name, employees.emp_name
FROM departments
```

LEFT JOIN employees ON departments.dept_id = employees.dept_id;

SQL Group By

Theory Questions:

1. What is the GROUP BY clause in SQL? How is it used with aggregate functions?
   - The GROUP BY clause groups rows with identical column values, often used with aggregate functions like COUNT, SUM, AVG, MIN, and MAX to provide summary statistics for each group.
2. Explain the difference between GROUP BY and ORDER BY.
   - GROUP BY groups rows with the same values *before* applying aggregate functions. ORDER BY sorts the result set *after* data retrieval and aggregation.

LAB EXERCISES:

- Lab 1: Group employees by department and count the number of employees in each department using GROUP BY.
  SELECT departments.dept_name, COUNT(employees.emp_id) AS employee_count
  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. (Assuming you have added salary column in employees table)
  -- Alter the employees table to add a salary column
  ALTER TABLE employees
  ADD COLUMN salary DECIMAL(10, 2);

  -- Add some 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;

  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;

SQL Stored Procedure

Theory Questions:

1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?
   - A stored procedure is a precompiled set of SQL statements stored in the database. Unlike a standard SQL query, which is executed directly, a stored procedure is invoked by name, offering improved performance and reusability.
2. Explain the advantages of using stored procedures.
   - Stored procedures offer several advantages, including reduced network traffic (as the entire procedure is executed on the database server), improved performance (due to precompilation), enhanced security (by controlling data access), and code reusability.

LAB EXERCISES:

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

  ```
  --  The specifics of stored procedure syntax vary by database (MySQL, SQL Server, PostgreSQL, etc.)
  --  This is a general example for MySQL.

  DELIMITER //
  CREATE PROCEDURE GetEmployeesByDepartment(IN deptName VARCHAR(255))
  BEGIN
     SELECT e.emp_name
     FROM employees e
     JOIN departments d ON e.dept_id = d.dept_id
     WHERE d.dept_name = deptName;
  END //
  DELIMITER ;

  --  To call the stored procedure:
  CALL GetEmployeesByDepartment('Sales');
  ```

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

  ```
  --  General example for MySQL
  ```

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

--  To call the procedure:
CALL GetCourseDetails(101);
```

SQL View

Theory Questions:

1. What is a view in SQL, and how is it different from a table?
   - A view is a virtual table based on the result set of a SQL query. Unlike a table, which stores data physically, a view does not store data itself but provides a specific perspective on the underlying data in one or more tables.
2. Explain the advantages of using views in SQL databases.
   - Views offer advantages such as simplified queries, data security (by restricting access to certain data), data independence, and a consistent data representation, even if the underlying table structure changes.

LAB EXERCISES:

- Lab 1: Create a view to show all employees along with their department names.
```
CREATE VIEW employee_departments_view AS
SELECT e.emp_name, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;

-- To select from the view
SELECT * FROM employee_departments_view;
```

- Lab 2: Modify the view to exclude employees whose salaries are below $50,000.
```
--  This assumes you have a salary column in the employees table.
CREATE OR REPLACE VIEW employee_departments_view AS
SELECT e.emp_name, d.dept_name
FROM employees e
```

```
JOIN departments d ON e.dept_id = d.dept_id
WHERE e.salary >= 50000;
```

SQL Triggers

Theory Questions:

1. What is a trigger in SQL? Describe its types and when they are used.
   - A trigger is a special type of stored procedure that automatically executes in response to specific events on a database table.
   - Types of Triggers:
     - BEFORE: Executes before the triggering event.
     - AFTER: Executes after the triggering event.
     - INSTEAD OF: Executes instead of the triggering event (used with views).
   - Triggers are used for:
     - Auditing data changes.
     - Enforcing complex business rules.
     - Maintaining data integrity.
     - Automating tasks.
2. Explain the difference between INSERT, UPDATE, and DELETE triggers.
   - INSERT Trigger: активируется при добавлении новой записи в таблицу.
   - UPDATE Trigger: срабатывает при изменении существующей записи в таблице.
   - DELETE Trigger: срабатывает при удалении записи из таблицы.

LAB EXERCISES:

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

```
--  This is a general example for MySQL.  Adjust for your DB.

CREATE TABLE employee_log (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   emp_id INT,
   event_type VARCHAR(10),
   event_time TIMESTAMP
);

DELIMITER //
CREATE TRIGGER log_new_employee
AFTER INSERT ON employees
```

```
FOR EACH ROW
BEGIN
   INSERT INTO employee_log (emp_id, event_type, event_time)
   VALUES (NEW.emp_id, 'INSERT', NOW());
END //
DELIMITER ;
```

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

```
-- This assumes you have a last_modified column in the employees table
ALTER TABLE employees ADD COLUMN last_modified TIMESTAMP DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

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

Introduction to PL/SQL

Theory Questions:

1. What is PL/SQL, and how does it extend SQL's capabilities?
   - PL/SQL (Procedural Language/SQL) is Oracle's extension to standard SQL. It adds procedural programming constructs (like loops, conditions, and variables) to SQL, enabling the creation of more complex and sophisticated database applications. It allows you to write code, not just queries.
2. List and explain the benefits of using PL/SQL.
   - Benefits of PL/SQL:
     - Procedural Capabilities: Allows for complex logic.
     - Improved Performance: Reduces network traffic.
     - Modularity: Supports reusable code blocks.
     - Error Handling: Provides robust exception handling.
     - Security: Enhances data protection.

        - Integration with SQL: Seamlessly works with SQL statements.

LAB EXERCISES:

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

```
--  Oracle PL/SQL Example

DECLARE
   total_employees NUMBER;
BEGIN
   SELECT COUNT(*) INTO total_employees FROM employees;
   DBMS_OUTPUT.PUT_LINE('Total Employees: ' || total_employees);
END;
/
```

- Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.

```
-- Oracle PL/SQL
DECLARE
   total_sales NUMBER;
BEGIN
   SELECT SUM(order_amount) INTO total_sales FROM orders;  --  Assuming
'orders' table and 'order_amount' column
   IF total_sales IS NULL THEN
      DBMS_OUTPUT.PUT_LINE('No sales data available.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Total Sales: ' || total_sales);
   END IF;
END;
/
```

PL/SQL Control Structures

Theory Questions:

1. What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.
    - Control structures manage the flow of execution in PL/SQL.
        - IF-THEN: Executes code based on a condition.
        - LOOP: Repeats a sequence of statements.
2. How do control structures in PL/SQL help in writing complex queries?

    - They allow for conditional execution of SQL statements, iteration over data, and implementation of complex business logic that cannot be achieved with simple SQL queries.

LAB EXERCISES:

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

  ```
  -- Oracle PL/SQL
  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;  --  Example:  Get data for employee with ID 101

     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: ' ||
  employee_dept);
     END IF;
  END;
  /
  ```

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

  ```
  -- Oracle PL/SQL
  BEGIN
     FOR emp_record IN (SELECT emp_name FROM employees) LOOP
        DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.emp_name);
     END LOOP;
  END;
  /
  ```

SQL Cursors

Theory Questions:

1. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.
   - A cursor is a pointer to a memory area that holds the result set of a SELECT statement.
     - Implicit Cursor: Automatically created by Oracle for SQL statements.
     - Explicit Cursor: Declared and managed by the programmer for more control.
2. When would you use an explicit cursor over an implicit one?
   - Use explicit cursors when you need to:
     - Fetch rows one at a time.
     - Process each row individually.
     - Control cursor attributes (e.g., %ISOPEN, %FOUND, %NOTFOUND).
     - Perform updates or deletes on the fetched rows.

LAB EXERCISES:

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

```
--  Oracle PL/SQL
DECLARE
   CURSOR emp_cursor IS
      SELECT emp_name, dept_name, salary
      FROM employees e JOIN departments d ON e.dept_id = d.dept_id;
   v_emp_name VARCHAR2(255);
   v_dept_name VARCHAR2(255);
   v_salary NUMBER;
BEGIN
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO v_emp_name, v_dept_name, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name || ', Department: ' ||
v_dept_name || ', Salary: ' || v_salary);
   END LOOP;
   CLOSE emp_cursor;
END;
/
```

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

```
-- Oracle PL/SQL
DECLARE
  CURSOR course_cursor IS
    SELECT course_name FROM courses;
  v_course_name VARCHAR2(255);
BEGIN
  OPEN course_cursor;
  LOOP
    FETCH course_cursor INTO v_course_name;
    EXIT WHEN course_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Course: ' || v_course_name);
  END LOOP;
  CLOSE course_cursor;
END;
/
```

Rollback and Commit Savepoint

Theory Questions:

1. Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?
   - SAVEPOINT: Marks a point within a transaction to which you can later roll back.
   - ROLLBACK TO SAVEPOINT: Undoes changes *since* the savepoint, without affecting earlier parts of the transaction.
   - ROLLBACK: Without a savepoint, it undoes the *entire* transaction.
   - COMMIT: Saves *all* changes in the transaction, including those before and after any savepoints.
2. When is it useful to use savepoints in a database transaction?
   - Savepoints are useful for:
     - Breaking down long transactions into smaller, manageable parts.
     - Partially undoing a transaction.
     - Implementing complex error recovery procedures.
     - Allowing for conditional transaction processing.

LAB EXERCISES:

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

```
-- Oracle PL/SQL
BEGIN
   INSERT INTO courses (course_id, course_name, course_duration) VALUES (301,
'Physics', '4 weeks');
   SAVEPOINT save1;
   INSERT INTO courses (course_id, course_name, course_duration) VALUES (302,
'Chemistry', '4 weeks');
   ROLLBACK TO save1;  -- Undoes the insert of course 302
   --  Course 301 is still inserted, because it happened *before* the savepoint.
   COMMIT;  --  Commits the insertion of course 301
END;
/
```

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

```
-- Oracle PL/SQL
BEGIN
   INSERT INTO courses (course_id, course_name, course_duration) VALUES (401,
'Biology', '4 weeks');
   SAVEPOINT save1;
   INSERT INTO courses (course_id, course_name, course_duration) VALUES (402,
'Geology', '4 weeks');
   COMMIT;  -- Commits the insert of 401.
   ROLLBACK TO save1;
   --  The insert of 402 is rolled back, but the insert of 401 is *not*, because it was
committed.
END;
/
```