**Name: Arjun Prajapati Course: Python Full Stack**

**Module 4 – Introduction to DBMS**

Introduction to SQL

**Theory Questions**:

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

SQL (Structured Query Language) is a standardized programming language designed for managing and manipulating relational databases. It enables users to perform various operations such as querying data, creating and modifying database structures, and managing access permissions. SQL's widespread adoption across relational database management systems (RDBMS) like MySQL, PostgreSQL, Oracle, and SQL Server highlights its importance in database management.

1. Explain the difference between DBMS and RDBMS.

A Database Management System (DBMS) is software that manages databases, allowing users to store, retrieve, and manipulate data. However, DBMS typically supports a single user and may not enforce data integrity constraints. In contrast, a Relational Database Management System (RDBMS) is a type of DBMS that stores data in a structured, tabular form using rows and columns. RDBMSs support multiple users and enforce data integrity constraints, ensuring accuracy and consistency. They also use SQL as the standard language for managing and querying data.

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

SQL serves as the primary interface for interacting with relational databases. It allows users to:

**Data Querying:** Retrieve specific data using SELECT statements with various filtering and sorting options.

**Data Manipulation:** Insert new records, update existing ones, and delete records as needed.

**Schema Definition:** Define and modify database structures, such as tables and relationships, using Data Definition Language (DDL) commands like CREATE, ALTER, and DROP.

**Access Control:** Manage user permissions and security settings to control access to sensitive data.

1. What are the key features of SQL?

SQL offers several key features that make it effective for database management:

**Declarative Language:** Users specify what data to retrieve or manipulate, without detailing how to perform these operations.

**Portability:** SQL is standardized, allowing queries to run on different RDBMS platforms with minimal adjustments.

**Scalability:** It can handle large volumes of data, supporting both small-scale and enterprise-level applications.

**Security:** SQL provides mechanisms to define user roles and permissions, ensuring controlled access to data.

**Transaction Management:** It supports transactions, ensuring data integrity and consistency through features like ACID (Atomicity, Consistency, Isolation, Durability) properties.

**Integration:** SQL integrates seamlessly with various programming languages, enabling dynamic data-driven applications.

**LAB EXERCISES**:

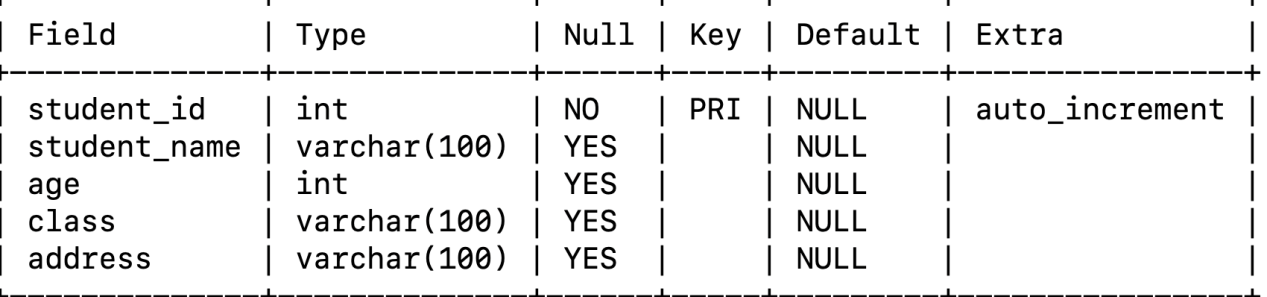


**Lab 1**: Create a new database named school\_db and a table called students with the

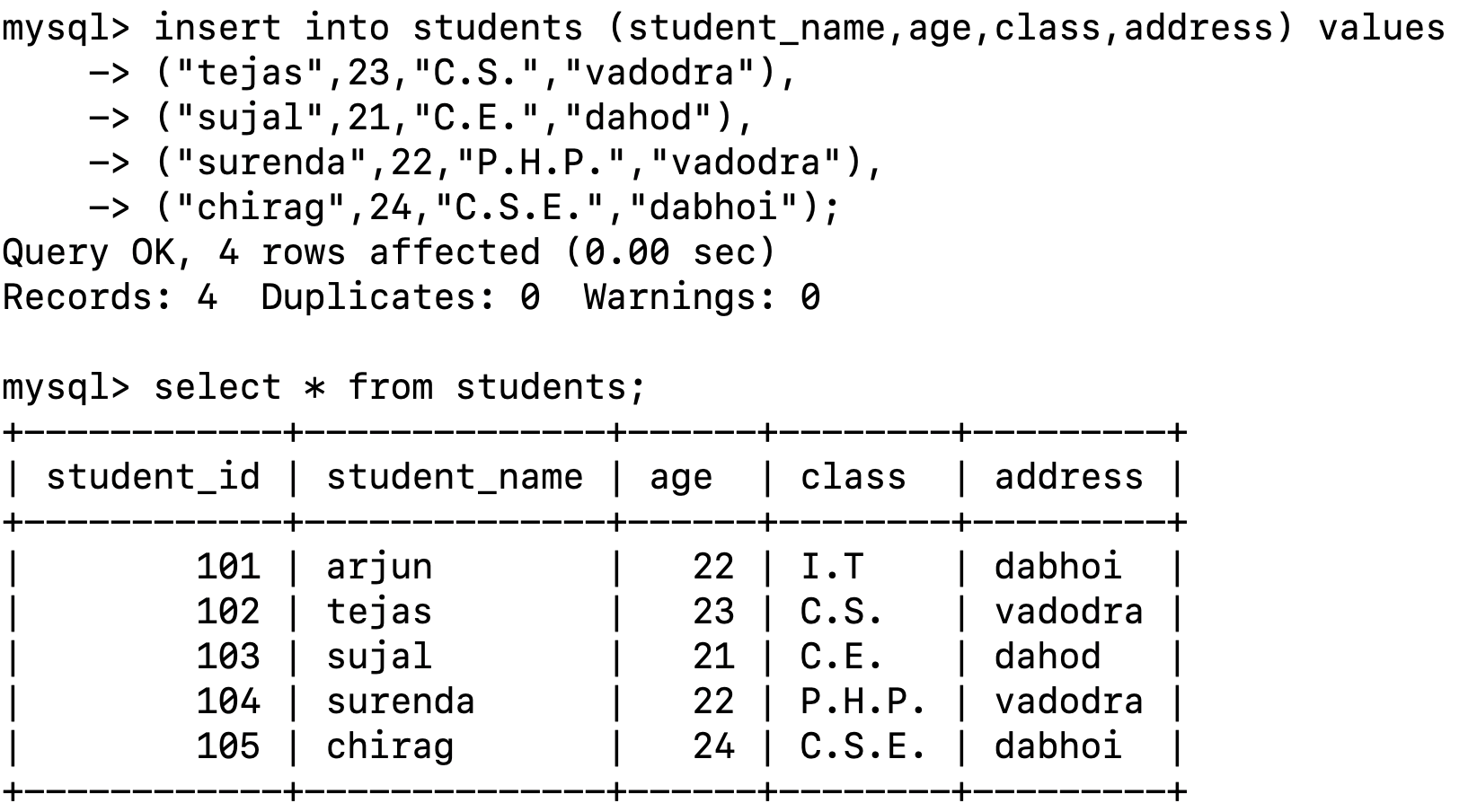
following columns: student\_id, student\_name, age, class, and address.

create database school\_db;

create table students(student\_id int not null primary key,student\_name varchar(100),age int,class varchar(100),address varchar(100));



**Lab 2**: Insert five records into the students table and retrieve all records using the SELECT statement.



*2.* SQL Syntax

**Theory Questions**:

1. What are the basic components of SQL syntax?

SQL syntax comprises several key elements that define how SQL statements are structured:

**Keywords:** Reserved words that have specific meanings in SQL, such as SELECT, FROM, WHERE, INSERT, UPDATE, and DELETE.

**Identifiers:** Names of database objects like tables, columns, and schemas. Identifiers can be delimited (enclosed in double quotes) to allow the use of reserved keywords or special characters.

**Clauses:** Components of SQL statements that perform specific functions. For example, the SELECT clause specifies columns to retrieve, and the FROM clause specifies tables to query.

**Expressions:** Constructs that produce values, such as arithmetic calculations or string concatenations.

**Predicates:** Conditions used in WHERE or HAVING clauses to filter data, evaluating to true, false, or unknown.

**Queries:** Statements that retrieve data based on specific criteria.

**Statements:** Commands that perform actions like data retrieval, modification, or schema changes.

**Semicolons:** Statement terminators that denote the end of an SQL statement, though their use can vary by SQL dialect.

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

The general structure of an SQL SELECT statement is as follows:

SELECT column1, column2, ...FROM table\_name WHERE condition ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

**SELECT clause:** Specifies the columns to retrieve. For example, SELECT first\_name, last\_name.

**FROM clause:** Identifies the table from which to retrieve data. For example, FROM employees.

**WHERE clause:** Optional; filters records based on specified conditions. For example, WHERE department = 'Sales'.

**ORDER BY clause:** Optional; sorts the result set by one or more columns, either ascending (ASC) or descending (DESC). For example, ORDER BY last\_name ASC.

This structure allows for flexible data retrieval and manipulation within a relational database.

1. Explain the role of clauses in SQL statements.

Clauses in SQL statements serve specific purposes to define and control the operations performed on the database. Key clauses include:

**SELECT:** Identifies the columns to retrieve from one or more tables.

**FROM:** Specifies the table(s) from which to retrieve data.

**WHERE:** Filters records based on specified conditions, limiting the rows returned.

**GROUP BY:** Groups rows sharing a property, often used with aggregate functions like COUNT, SUM, or AVG.

**HAVING:** Filters groups based on specified conditions, similar to WHERE but applied after grouping.

**ORDER BY:** Sorts the result set by one or more columns, either ascending or descending.

**JOIN:** Combines rows from two or more tables based on related columns.

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

**UPDATE:** Modifies existing records in a table.

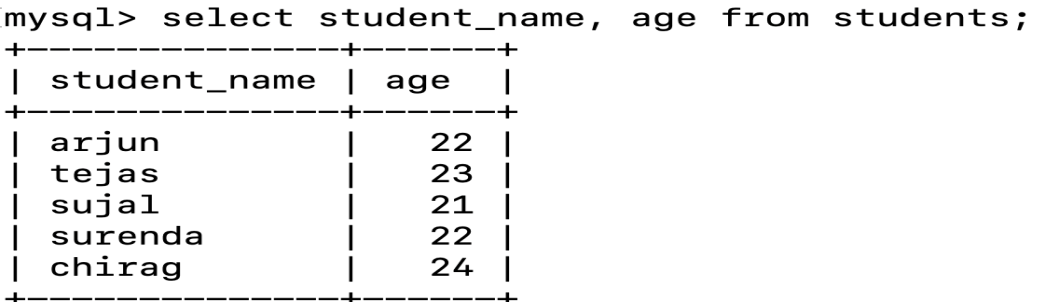
**DELETE:** Removes records from a table.

Each clause plays a vital role in shaping SQL statements to perform precise data operations, ensuring efficient and accurate database management.

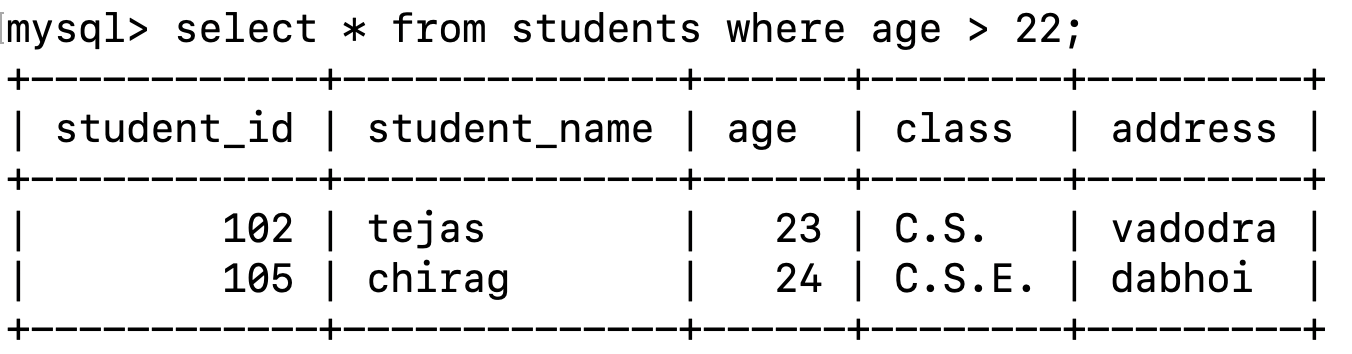
**LAB EXERCISES**:



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



**Lab 2**: Write SQL queries to retrieve all students whose age is greater than 22.



*3.* SQL Constraints

**Theory Questions**:

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

**SQL Constraints**: Constraints in SQL are rules applied to columns or tables to ensure the accuracy and reliability of the data within a database. They limit the type of data that can be inserted, updated, or deleted, maintaining data integrity.

**Types of Constraints**:

**NOT NULL**: Ensures that a column cannot have NULL values.

**UNIQUE**: Guarantees that all values in a column are distinct.

**PRIMARY KEY**: Uniquely identifies each record in a table; combines NOT NULL and UNIQUE properties.

**FOREIGN KEY**: Maintains referential integrity by ensuring that a value in one table matches a value in another table's primary key.

**CHECK**: Ensures that all values in a column satisfy a specific condition.

**DEFAULT**: Provides a default value for a column when none is specified.

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

**PRIMARY KEY**: Ensures that each record in a table is uniquely identifiable. A table can have only one primary key, which may consist of one or multiple columns.

**FOREIGN KEY**: Establishes a link between two tables by ensuring that the value in a column (or a combination of columns) matches a value in the referenced table's primary key. This maintains referential integrity between the tables.

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

**NOT NULL**: Prevents NULL values from being entered into a column, ensuring that every record has a value for that column.

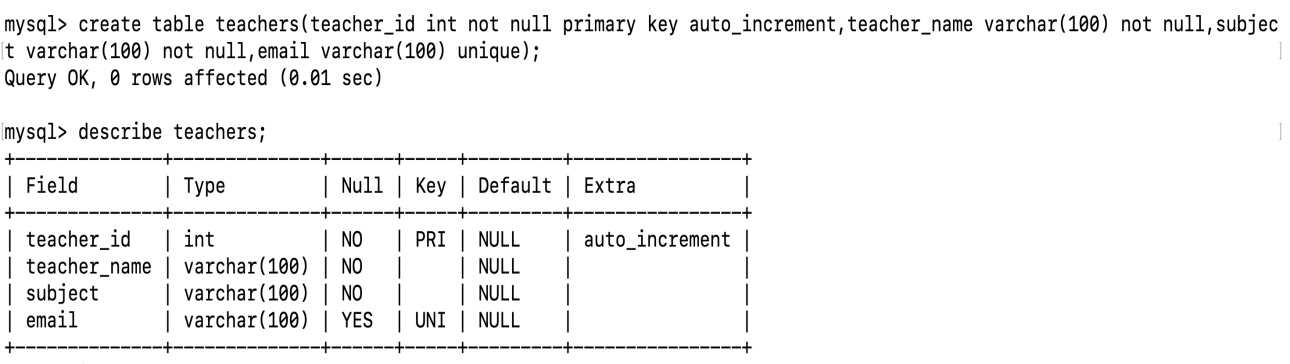
**UNIQUE**: Ensures that all values in a column are distinct, preventing duplicate entries. Unlike the PRIMARY KEY constraint, UNIQUE constraints allow for multiple NULL values, depending on the database system.

**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).



**Lab 2**: Implement a FOREIGN KEY constraint to relate the teacher\_id from the

teachers table with the students table.

ALTER TABLE students

ADD CONSTRAINT fk\_teacher\_id FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id) ON DELETE CASCADE;

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

**Theory Questions**:

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

SQL Data Definition Language (DDL) is a subset of SQL that is used to define and manage database structures, such as tables, indexes, views, and schemas. DDL statements allow you to create, modify, and delete database objects. These commands do not manipulate the actual data within the tables but focus on the structure of the database itself. The key DDL commands include:

CREATE: To create new database objects like tables, views, indexes, etc.

ALTER: To modify an existing database object.

DROP: To delete a database object.

TRUNCATE: To remove all records from a table but retain the structure.

1. Explain the CREATE command and its syntax.

The CREATE command is used to create new database objects such as tables, views, indexes, or databases. Specifically, in the context of tables, the CREATE TABLE command defines the structure of a table, including its columns and associated constraints.

#### Syntax for Creating a Table:

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

[table\_constraints]

);

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

### Purpose of Specifying Data Types and Constraints During Table Creation:

**Data Types**: Define the type of data a column can hold (e.g., INT, VARCHAR, DATE), ensuring only valid data is stored in each column.

**Constraints**: Enforce rules on the data (e.g., NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE) to maintain data integrity, consistency, and relationships between tables. They prevent invalid or inconsistent data from being entered and help optimize performance.

**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(100) NOT NULL,course\_credits INT NOT NULL);

**Lab 2**: Use the CREATE command to create a database university\_db.

CREATE DATABASE university\_db;

5. ALTER Command

**Theory Questions**:

1. What is the use of the ALTER command in SQL?

The ALTER command in SQL is used to modify the structure of an existing database object, such as a table. It allows you to perform actions like adding, deleting, or modifying columns, as well as altering constraints or renaming objects within a table. The ALTER command is essential when you need to make changes to the structure of a table without affecting the existing data.

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

**Add a column**:

ALTER TABLE table\_name ADD column\_name datatype;

**Modify a column**:

ALTER TABLE table\_name MODIFY column\_name new\_datatype;

**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 course\_duration VARCHAR(50);



**Lab 2**: 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?

The DROP command is used to permanently remove database objects like tables, views, or indexes. When you drop a table, both the table structure and all of its data are deleted irreversibly.

**Example**:

DROP TABLE table\_name;

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

**Data Loss**: All data in the table is permanently deleted.

**Structure Removal**: The table and its schema (columns, constraints) are removed.

**Dependency Impact**: Foreign keys, indexes, views, and other objects dependent on the table may break or be removed.

**Irreversible**: Once dropped, the table and its data cannot be recovered unless you have a backup.

**LAB EXERCISES**:



**Lab 1**: Drop the teachers table from the school\_db database.

DROP TABLE school\_db.teachers;



**Lab 2**: Drop the students table from the school\_db database and verify that the table has been removed.

DROP TABLE school\_db.students;

7. Data Manipulation Language (DML)

**Theory Questions**:

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

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

INSERT INTO table\_name (columns) VALUES (values);

**UPDATE**: Modifies existing records in a table.

UPDATE table\_name SET column = value WHERE condition;

**DELETE**: Removes records from a table.

DELETE FROM table\_name WHERE condition;

1. What is the importance of the WHERE clause in UPDATE and DELETE operations?

**In UPDATE**: The WHERE clause ensures only specific rows are updated. Without it, all rows in the table are updated.

**In DELETE**: The WHERE clause ensures only specific rows are deleted. Without it, all rows in the table are deleted.

The WHERE clause prevents unintended changes or data loss by targeting specific rows.

**LAB EXERCISES**:



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

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

VALUES (1, 'Mathematics 101', 3, '3 months');

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

VALUES (2, 'History 201', 4, '1 semester');

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

VALUES (3, 'Computer Science 301', 5, '1 semester');



**Lab 2**: Update the course duration of a specific course using the UPDATE command.

UPDATE courses SET course\_duration = '4 months' WHERE course\_id = 1;



**Lab 3**: Delete a course with a specific course\_id from the courses table using the DELETE command.

DELETE FROM courses WHERE course\_id = 2;

8. Data Query Language (DQL)

**Theory Questions**:

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

The SELECT statement in SQL is used to retrieve data from one or more tables in a database. It allows you to specify the columns you want to retrieve and apply conditions to filter or sort the results.

**Basic Syntax**:

SELECT column1, column2, ...FROM table\_name WHERE condition;

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

ORDER BY **Clause**:  
The ORDER BY clause is used to sort the result set of a query. You can specify one or more columns by which to sort the data, and you can choose whether to sort in ascending (ASC) or descending (DESC) order.

**Syntax**:

SELECT column1, column2, ...FROM table\_name WHERE condition ORDER BY column\_name [ASC|DESC];

**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 ORDER BY course\_duration DESC LIMIT 2;

9. Data Control Language (DCL)

**Theory Questions**:

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

**GRANT**: The GRANT command is used to give specific privileges (permissions) to users or roles in a database. These privileges allow the users to perform certain operations like selecting, inserting, updating, or deleting data in the database.

**REVOKE**: The REVOKE command is used to remove or revoke previously granted privileges from users or roles. This restricts their ability to perform certain operations on the database.

1. How do you manage privileges using these commands?

**Using GRANT**: The GRANT command is used to assign specific privileges to a user or role. These privileges include permissions like SELECT, INSERT, UPDATE, DELETE, EXECUTE, etc.

**Syntax**:

GRANT privilege\_type ON object TO user;

**Using REVOKE**: The REVOKE command is used to remove privileges that were previously granted to a user or role. After revoking the privileges, the user will no longer be able to perform those actions on the specified object.

**Syntax**:

REVOKE privilege\_type ON object FROM user;

**LAB EXERCISES**:



**Lab 1**: Create two new users user1 and user2 and grant user1 permission to SELECT

from the courses table.

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';

CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

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

FLUSH PRIVILEGES;

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

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

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

FLUSH PRIVILEGES;

10. Transaction Control Language (TCL)

**Theory Questions**:

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

**COMMIT**: The COMMIT command is used to **save all the changes** made during the current transaction to the database permanently. Once a COMMIT is issued, the changes become part of the database, and they cannot be undone. It marks the end of a successful transaction.

**Purpose**:

It finalizes the changes made during the transaction.

Ensures that all changes are saved permanently and are visible to other users.

**ROLLBACK**: The ROLLBACK command is used to **undo** all changes made during the current transaction. It reverts the database back to its state before the transaction started, effectively discarding any changes made since the last COMMIT or BEGIN TRANSACTION.

**Purpose**:

It helps to cancel a transaction if there is an error or if the operation should not be completed.

Ensures that no partial or incomplete changes are saved to the database.

2. Explain how transactions are managed in SQL databases.

**Transactions** in SQL are managed using four key properties, known as the **ACID** properties:

**Atomicity**:Ensures that a transaction is treated as a single, indivisible unit. Either all operations in a transaction are executed, or none are. If any operation fails, the entire transaction is rolled back.

**Consistency**:Guarantees that a transaction brings the database from one valid state to another. The integrity constraints are always satisfied before and after the transaction.

**Isolation**:Ensures that the operations of a transaction are isolated from other transactions. The changes made by a transaction are not visible to other transactions until the transaction is committed.

**Durability**:Ensures that once a transaction is committed, its changes are permanent, even in the event of a system crash.

**LAB EXERCISES**:



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

BEGIN TRANSACTION;

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

VALUES (101, 'Mathematics 101', 3, '3 months');

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

VALUES (102, 'History 101', 4, '1 semester');

COMMIT;

**Lab 2**: Insert additional rows, then use ROLLBACK to undo the last insert operation.

BEGIN TRANSACTION;

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

VALUES (103, 'Computer Science 101', 5, '1 semester');

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

VALUES (104, 'Physics 101', 3, '3 months');

ROLLBACK;



**Lab 3**: Create a SAVEPOINT before updating the courses table, and use it to roll back

specific changes.

BEGIN TRANSACTION;

SAVEPOINT before\_update;

UPDATE courses

SET course\_name = 'Advanced Mathematics 101'

WHERE course\_id = 101;

ROLLBACK TO SAVEPOINT before\_update;

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?

A **JOIN** combines rows from two or more tables based on a related column. It's used to retrieve data from multiple tables in a single query.

#### Types of Joins:

**INNER JOIN**:Returns only rows where there is a match in both tables.

SELECT \* FROM table1 INNER JOIN table2 ON table1.id = table2.id;

**LEFT JOIN (LEFT OUTER JOIN)**:Returns all rows from the left table and matching rows from the right table. If no match, returns NULL for right table columns.

SELECT \* FROM table1 LEFT JOIN table2 ON table1.id = table2.id;

**RIGHT JOIN (RIGHT OUTER JOIN)**:Returns all rows from the right table and matching rows from the left table. If no match, returns NULL for left table columns.

SELECT \* FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;

**FULL OUTER JOIN**:Returns all rows from both tables, with NULL where there’s no match.

SELECT \* FROM table1 FULL OUTER JOIN table2 ON table1.id = table2.id;

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

Joins combine data from multiple tables using a related column, often a **foreign key** in one table linking to the **primary key** in another. Based on the type of join, it returns matched rows or all rows with NULL for missing data.

**Example**:  
Combining students and courses tables on course\_id using an INNER JOIN:

SELECT students.student\_id, courses.course\_nameFROM studentsINNER JOIN courses ON students.course\_id = courses.course\_id;

This returns only students with a course.

**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 (department\_id INT PRIMARY KEY,department\_name VARCHAR(100) NOT NULL);

INSERT INTO departments (department\_id, department\_name)VALUES (1, 'HR'), (2, 'IT'), (3, 'Finance');

CREATE TABLE employees (employee\_id INT PRIMARYKEY,employee\_name VARCHAR(100) NOT NULL,department\_id INT,FOREIGN KEY (department\_id) REFERENCES departments(department\_id));

INSERT INTO employees (employee\_id, employee\_name, department\_id)VALUES (1, 'John Doe', 1), (2, 'Jane Smith', 2), (3, 'Alice Johnson', 3);

SELECT employees.employee\_name, departments.department\_name FROM employees INNER JOIN departments ON employees.department\_id = departments.department\_id;

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

INSERT INTO departments (department\_id, department\_name)VALUES (4, 'Marketing');

SELECT departments.department\_name, employees.employee\_name FROM departments LEFT JOIN employees ON departments.department\_id = employees.department\_id;

12. SQL Group By

**Theory Questions**:

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

Groups rows by one or more columns.

Used with aggregate functions (e.g., COUNT(), SUM(), AVG()) to calculate values for each group.

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

GROUP BY**:** Groups rows into summary results.

ORDER BY**:** Sorts the result set based on one or more columns (ascending or descending).

**LAB EXERCISES**:



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

SELECT department, COUNT(\*) AS number\_of\_employees

FROM employees

GROUP BY department;



**Lab 2**: Use the AVG aggregate function to find the average salary of employees in each department.

SELECT department, AVG(salary) AS average\_salary

FROM employees

GROUP BY department;

13. 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 that can be executed multiple times.

**Difference:** A standard SQL query is a single, one-time statement, while a stored procedure is reusable, can accept parameters, and may include logic.

1. Explain the advantages of using stored procedures.

**Reusability:** Can be executed multiple times.

**Performance:** Precompiled for faster execution.

**Security:** Controls access to underlying data.

**Maintainability:** Easier to manage business logic.

**Error Handling:** Allows for structured error management.

**Reduced Network Traffic:** Minimizes client-server communication by executing multiple statements in one call.

**LAB EXERCISES**:



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

DELIMITER //

CREATE PROCEDURE GetEmployeesByDepartment (IN dept\_name VARCHAR(50))

BEGIN

SELECT \* FROM employees WHERE department = dept\_name;

END //

DELIMITER ;



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

DELIMITER //

CREATE PROCEDURE GetCourseDetails (IN course\_id INT)

BEGIN

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

END //

DELIMITER ;

14. SQL View

**Theory Questions**:

1. What is a view in SQL, and how is it different from a table?

**View:**A **view** in SQL is a virtual table created by a query that selects data from one or more tables. It does not store data itself but provides a way to view the result of a query as if it were a table.

A view can simplify complex queries, provide a customized representation of the data, and improve security by limiting access to specific columns or rows.

**Difference from a Table:**

**Table:** A table physically stores data in the database.

**View:** A view does not store data; it only shows the result of a query dynamically when accessed.

1. Explain the advantages of using views in SQL databases.

**Simplify Complex Queries:**Views can encapsulate complex joins, unions, and aggregations, making queries simpler and easier to manage.

**Security:**Views can restrict access to specific columns or rows of a table. For example, you can create a view that shows only specific data and grant users permission to access the view rather than the underlying table.

**Data Abstraction:**Views provide an abstraction layer by hiding the underlying complexity of the data structure, allowing users to focus on the data without worrying about its organization.

**Reusability:**Once a view is created, it can be reused in different queries, reducing code duplication.

**Consistent Data Representation:**Views can ensure that the same query logic is consistently applied when accessing data, helping to avoid discrepancies or errors in data retrieval.

**Read-Only Views (optional):**Some views are read-only, which means they can be used for querying data without the risk of modifying the underlying tables.

**LAB EXERCISES**:



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

CREATE VIEW EmployeeWithDepartment AS

SELECT employees.employee\_id, employees.name, employees.salary, departments.department\_name

FROM employees

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



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

CREATE OR REPLACE VIEW EmployeeWithDepartment AS

SELECT employees.employee\_id, employees.name, employees.salary, departments.department\_name

FROM employees

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

WHERE employees.salary >= 50000;

15. SQL Triggers

**Theory Questions**:

1. What is a trigger in SQL? Describe its types and when they are used.

A **trigger** is a set of SQL instructions that automatically execute when a specific event (INSERT, UPDATE, or DELETE) occurs on a table. It is used for tasks like data validation, auditing, or cascading updates.

**Types of Triggers:**

**BEFORE Trigger:** Executes before the data modification.

**AFTER Trigger:** Executes after the data modification.

**INSTEAD OF Trigger:** Executes in place of the data modification (often used with views).

1. Explain the difference between INSERT, UPDATE, and DELETE triggers.

INSERT **Trigger:** Fires when a new row is added to the table.

UPDATE **Trigger:** Fires when an existing row is modified.

DELETE **Trigger:** Fires when a row is deleted from the table.

Each trigger type is used to handle different actions depending on whether data is inserted, updated, or deleted.

**LAB EXERCISES**:



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

CREATE TRIGGER LogEmployeeInsertion AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_audit\_log (action, employee\_id, timestamp)VALUES ('INSERT', NEW.employee\_id, NOW());

END;



**Lab 2**: Create a trigger to update the last\_modified timestamp whenever an employee

record is updated.

CREATE TRIGGER UpdateLastModifiedTimestamp

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

SET NEW.last\_modified = NOW();

END;

16. Introduction to PL/SQL

**Theory Questions**:

1. What is PL/SQL, and how does it extend SQL's capabilities?

**PL/SQL** is Oracle's procedural extension to SQL, allowing you to write programs that include variables, loops, conditions, and exception handling, alongside SQL queries. It extends SQL by adding procedural features, enabling complex logic, error handling, and performance optimizations within the database.

1. List and explain the benefits of using PL/SQL.

**Improved Performance:** Reduces network traffic by executing multiple SQL operations in a single call.

**Reusability:** Code can be reused through stored procedures and functions.

**Error Handling:** Offers advanced exception handling.

**Complex Logic:** Supports loops, conditions, and variables for implementing business logic.

**Security:** Encapsulates logic in the database, reducing exposure to the client.

**Maintainability:** Promotes modular code that's easier to maintain and update.

**Portability:** PL/SQL works across all Oracle platforms.

**Atomic Transactions:** Supports grouping multiple operations into a single transaction for

consistency.

**LAB EXERCISES**:



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

DECLARE

total\_employees NUMBER;

BEGIN

-- Query to get the total number of employees

SELECT COUNT(\*) INTO total\_employees FROM employees;

-- Print the result

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.

DECLARE

total\_sales NUMBER;

BEGIN

-- Query to calculate the total sales from the orders table

SELECT SUM(order\_amount) INTO total\_sales FROM orders;

-- Print the result

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.

**Control structures** allow for decision-making and looping in PL/SQL.

#### ****IF-THEN:****

Executes a set of statements if a condition is true.

IF condition THEN

-- Statements

ELSE

--Alternative statements

END IF;

#### ****LOOP:****

Repeats a set of statements indefinitely or based on a condition.

-- Statements

EXIT WHEN condition; -- Exit condition

END LOOP;

1. How do control structures in PL/SQL help in writing complex queries?

Control structures allow you to:

**Make decisions** (e.g., using IF-THEN) to execute different queries or logic based on conditions.

**Repeat actions** (e.g., using LOOP) to handle repetitive tasks or iterate through data, making complex queries easier to manage and more flexible.

**LAB EXERCISES**:



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

DECLARE

v\_department VARCHAR2(50);

BEGIN

-- Assume we are checking the department of employee with ID 3 (Charlie)

SELECT department INTO v\_department

FROM employees

WHERE employee\_id = 3;

-- Use IF-THEN to check the department

IF v\_department = 'HR' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee is in the HR department.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee is not in the HR department.');

END IF;

END;

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

DECLARE

CURSOR emp\_cursor IS

SELECT name FROM employees;

BEGIN

-- Use a FOR LOOP to iterate through the employee records

FOR emp IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || 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.

A **cursor** in PL/SQL is a pointer to the result set of a query, used to process multiple rows of data.

#### ****Implicit Cursor:****

**Definition:** Automatically created by Oracle for single-row queries or DML operations (e.g., SELECT INTO, INSERT, UPDATE, DELETE).

**Characteristics:** Automatically handled, no need for explicit opening or closing.

#### ****Explicit Cursor:****

**Definition:** Defined by the programmer for queries that return multiple rows, used for row-by-row processing.

**Characteristics:** Requires manual handling (open, fetch, close).

1. When would you use an explicit cursor over an implicit one?

**Explicit Cursor:** Use when you need to process multiple rows, perform complex operations, or manage multiple fetches.

**Implicit Cursor:** Use for single-row queries or DML statements where you don't need row-by-row control.

**LAB EXERCISES**:



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

DECLARE

-- Declare the cursor to retrieve employee details

CURSOR emp\_cursor IS

SELECT employee\_id, name, department, salary FROM employees;

-- Declare variables to hold fetched data

v\_employee\_id employees.employee\_id%TYPE;

v\_name employees.name%TYPE;

v\_department employees.department%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

-- Open the cursor

OPEN emp\_cursor;

-- Fetch and display employee details

LOOP

FETCH emp\_cursor INTO v\_employee\_id, v\_name, v\_department, v\_salary;

-- Exit the loop when no more rows are fetched

EXIT

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

DECLARE

-- Declare the cursor to retrieve course details

CURSOR course\_cursor IS

SELECT course\_id, course\_name FROM courses;

-- Declare variables to hold fetched course data

v\_course\_id courses.course\_id%TYPE;

v\_course\_name courses.course\_name%TYPE;

BEGIN

-- Open the cursor

OPEN course\_cursor;

-- Fetch and display course details

LOOP

FETCH course\_cursor INTO v\_course\_id, v\_course\_name;

-- Exit the loop when no more rows are fetched

EXIT WHEN course\_cursor%NOTFOUND;

-- Display course details

DBMS\_OUTPUT.PUT\_LINE('Course ID: ' || v\_course\_id ||

', Course Name: ' || v\_course\_name);

END LOOP;

-- Close the cursor

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?

A **SAVEPOINT** marks a point in a transaction, allowing you to roll back to it instead of rolling back the entire transaction.

**ROLLBACK TO SAVEPOINT:** Rolls back changes made after the savepoint.

**COMMIT:** Finalizes the transaction, making all changes permanent.

**ROLLBACK (without SAVEPOINT):** Rolls back the entire transaction.

1. When is it useful to use savepoints in a database transaction?

**Partial Rollback:** Undo only part of a transaction, not the entire transaction.

**Error Handling:** Roll back to a savepoint when an error occurs after making some changes, without affecting earlier operations.

**LAB EXERCISES**:



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

BEGIN;

-- Insert first record

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

VALUES (1, 'John', 'HR', 50000);

-- Create a savepoint

SAVEPOINT before\_second\_insert;

-- Insert second record

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

VALUES (2, 'Alice', 'IT', 60000);

-- Rollback to savepoint (undo the second insert)

ROLLBACK TO SAVEPOINT before\_second\_insert;

-- Commit the first insert

COMMIT;

END;



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

BEGIN;

-- Insert first record

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

VALUES (1, 'John', 'HR', 50000);

-- Create a savepoint

SAVEPOINT before\_second\_insert;

-- Insert second record

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

VALUES (2, 'Alice', 'IT', 60000);

-- Commit the first part of the transaction (first insert)

COMMIT;

-- Insert third record

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

VALUES (3, 'Bob', 'Marketing', 55000);

-- Rollback to the savepoint (undo the third insert)

ROLLBACK TO SAVEPOINT before\_second\_insert;

END;

**EXTRA LAB PRACTISE FOR DATABASE CONCEPTS**

*1.* Introduction to SQL

**LAB EXERCISES**:



**Lab 3**: Create a database called library\_db and a table books with columns: book\_id,

title, author, publisher, year\_of\_publication, and price. Insert five records into

the table.



**Lab 4**: Create a table members in library\_db with columns: member\_id, member\_name,

date\_of\_membership, and email. Insert five records into this table.

*2.* SQL Syntax

**LAB EXERCISES**:



**Lab 3**: Retrieve all members who joined the library before 2022. Use appropriate SQL syntax

with WHERE and ORDER BY.



**Lab 4**: Write SQL queries to display the titles of books published by a specific author. Sort the

results by year\_of\_publication in descending order.*3.* SQL Constraints

**LAB EXERCISES**:



**Lab 3**: Add a CHECK constraint to ensure that the price of books in the books table is

greater than 0.



**Lab 4**: Modify the members table to add a UNIQUE constraint on the email column,

ensuring that each member has a unique email address.

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

**LAB EXERCISES**:



**Lab 3**: Create a table authors with the following columns: author\_id, first\_name,

last\_name, and country. Set author\_id as the primary key.



**Lab 4**: Create a table publishers with columns: publisher\_id, publisher\_name,

contact\_number, and address. Set publisher\_id as the primary key and

contact\_number as unique.

*5.* ALTER Command

**LAB EXERCISES**:



**Lab 3**: Add a new column genre to the books table. Update the genre for all existing

records.



**Lab 4**: Modify the members table to increase the length of the email column to 100

characters.

*6.* DROP Command

**LAB EXERCISES**:



**Lab 3**: Drop the publishers table from the database after verifying its structure.



**Lab 4**: Create a backup of the members table and then drop the original members table.*7.* Data Manipulation Language (DML)

**LAB EXERCISES**:



**Lab 4**: Insert three new authors into the authors table, then update the last name of one of

the authors.



**Lab 5**: Delete a book from the books table where the price is higher than $100.

*8.* UPDATE Command

**LAB EXERCISES**:



**Lab 3**: Update the year\_of\_publication of a book with a specific book\_id.



**Lab 4**: Increase the price of all books published before 2015 by 10%.

*9.* DELETE Command

**LAB EXERCISES**:



**Lab 3**: Remove all members who joined before 2020 from the members table.



**Lab 4**: Delete all books that have a NULL value in the author column.

*10.* Data Query Language (DQL)

**LAB EXERCISES**:



**Lab 4**: Write a query to retrieve all books with price between $50 and $100.



**Lab 5**: Retrieve the list of books sorted by author in ascending order and limit the results

to the top 3 entries.

*11.* Data Control Language (DCL)

**LAB EXERCISES**:



**Lab 3**: Grant SELECT permission to a user named librarian on the books table.



**Lab 4**: Grant INSERT and UPDATE permissions to the user admin on the members table.*12.* REVOKE Command

**LAB EXERCISES**:



**Lab 3**: Revoke the INSERT privilege from the user librarian on the books table.



**Lab 4**: Revoke all permissions from user admin on the members table.

*13.* Transaction Control Language (TCL)

**LAB EXERCISES**:



**Lab 3**: Use COMMIT after inserting multiple records into the books table, then make another

insertion and perform a ROLLBACK.



**Lab 4**: Set a SAVEPOINT before making updates to the members table, perform some

updates, and then roll back to the SAVEPOINT.

*14.* SQL Joins

**LAB EXERCISES**:



**Lab 3**: Perform an INNER JOIN between books and authors tables to display the title

of books and their respective authors' names.



**Lab 4**: Use a FULL OUTER JOIN to retrieve all records from the books and authors tables,

including those with no matching entries in the other table.

*15.* SQL Group By

**LAB EXERCISES**:



**Lab 3**: Group books by genre and display the total number of books in each genre.



**Lab 4**: Group members by the year they joined and find the number of members who joined

each year.

*16.* SQL Stored Procedure

**LAB EXERCISES**:



**Lab 3**: Write a stored procedure to retrieve all books by a particular author.



**Lab 4**: Write a stored procedure that takes book\_id as an argument and returns the price

of the book.

*17.* SQL View

**LAB EXERCISES**:



**Lab 3**: Create a view to show only the title, author, and price of books from the books

table.



**Lab 4**: Create a view to display members who joined before 2020.

*18.* SQL Trigger

**LAB EXERCISES**:



**Lab 3**: Create a trigger to automatically update the last\_modified timestamp of the

books table whenever a record is updated.



**Lab 4**: Create a trigger that inserts a log entry into a log\_changes table whenever a

DELETE operation is performed on the books table.

*19.* Introduction to PL/SQL

**LAB EXERCISES**:



**Lab 3**: Write a PL/SQL block to insert a new book into the books table and display a

confirmation message.



**Lab 4**: Write a PL/SQL block to display the total number of books in the books table.

*20.* PL/SQL Syntax

**LAB EXERCISES**:



**Lab 3**: Write a PL/SQL block to declare variables for book\_id and price, assign values, and

display the results.



**Lab 4**: Write a PL/SQL block using constants and perform arithmetic operations on book

prices.

*21.* PL/SQL Control Structures

**LAB EXERCISES**:



**Lab 3**: Write a PL/SQL block using IF-THEN-ELSE to check if a book's price is above $100

and print a message accordingly.



**Lab 4**: Use a FOR LOOP in PL/SQL to display the details of all books one by one.

*22.* SQL Cursors

**LAB EXERCISES**:



**Lab 3**: Write a PL/SQL block using an explicit cursor to fetch and display all records from the

members table.



**Lab 4**: Create a cursor to retrieve books by a particular author and display their titles.

*23.* Rollback and Commit Savepoint

**LAB EXERCISES**:



**Lab 3**: Perform a transaction that includes inserting a new member, setting a SAVEPOINT,

and rolling back to the savepoint after making updates.



**Lab 4**: Use COMMIT after successfully inserting multiple books into the books table, then use

ROLLBACK to undo a set of changes made after a savepoint.