Module 4 – Introduction to DBMS

Introduction to SQL

Theory Questions:

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

Ans. SQL means structure query language, it is a powerful language used to manage and interact with databases by enabling tasks like creating tables, inserting, updating, deleting, and retrieving data efficiently. It supports complex operations such as joining tables, filtering data with conditions, and performing aggregate calculations all while maintaining data integrity through constraints like primary keys and foreign keys. SQL also allows managing user permissions and ensuring secure access to the database. Its versatility and wide adoption in systems like MySQL, PostgreSQL, and SQL Server make it essential for backend development, data management, and analysis.

2. Explain the difference between DBMS and RDBMS.

Ans. DBMS (Database Management System) is software that manages databases, allowing users to create, retrieve, update, and delete data. It stores data as files without strict relationships between the data, making it suitable for small-scale applications. Examples include Microsoft Access and FileMaker.

RDBMS (Relational Database Management System) is an advanced type of DBMS that organizes data into tables with rows and columns, enforcing relationships using primary keys and foreign keys to maintain data integrity and eliminate redundancy. It supports ACID properties (Atomicity, Consistency, Isolation, Durability) for reliable transactions and is widely used in large, complex applications. Examples include MySQL, PostgreSQL, SQL Server, and Oracle.

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

Ans. SQL (Structured Query Language) is essential for managing relational databases as it allows users to define structures, manipulate data, and enforce relationships between tables. It helps create databases, add or modify records, and retrieve specific information using powerful queries with filtering, sorting, and joining capabilities. SQL also ensures data integrity through constraints like primary keys and foreign keys, supports transactions to maintain consistency, and manages user permissions for secure access making it the backbone of relational database management.

4. What are the key features of SQL?

Ans. The key features of SQL are:

**Data Definition** – SQL allows you to define and modify database structures using commands like CREATE, ALTER, and DROP.

**Data Manipulation** – It supports inserting, updating, deleting, and retrieving data with INSERT, UPDATE, DELETE, and SELECT commands.

**Data Querying** – SQL can fetch specific data using powerful queries with filtering (WHERE), sorting (ORDER BY), grouping (GROUP BY), and aggregation functions (COUNT, SUM, AVG).

**Data Control** – It manages user permissions and access with GRANT and REVOKE to ensure security.

**Transaction Control** – SQL supports transactions using commands like COMMIT, ROLLBACK, and SAVEPOINT, ensuring data consistency and recovery.

**Data Integrity** – It maintains data accuracy and reliability with constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK.

**Join Operations** – SQL allows combining data from multiple tables using JOIN operations (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN).

**Join Operations** – SQL allows combining data from multiple tables using JOIN operations (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN).

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 DATABASE school\_db;

USE school\_db;

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100),

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.

Ans.

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

VALUES

(1, 'Aarav Sharma', 14, '8A', '123 dindoli'),

(2, 'Ishita Verma', 15, '9B', '456 godadara'),

(3, 'Rohan Patel', 13, '7C', '789 rander'),

(4, 'Maya Kapoor', 14, '8B', '101 ring road'),

(5, 'Arjun Mehta', 16, '10A', '202 adajan');

SELECT \* FROM students;

2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?

Ans.

**Commands/Statements** – These are the core instructions that tell the database what to do. Common commands include SELECT for retrieving data, INSERT for adding new records, UPDATE for modifying existing data, DELETE for removing data, and CREATE or DROP for creating or deleting tables and databases.

**Clauses** – Clauses provide more control over commands. For example, WHERE filters rows based on a condition, ORDER BY sorts the results, GROUP BY groups rows with similar values, and HAVING filters the grouped data. Clauses help refine SQL queries for more precise outputs.

**Expressions** – An expression is a combination of columns, values, and operators that produces a single value. For example, age > 18 checks if a student is older than 18, while price \* quantity calculates a total cost. Expressions are essential for filtering, calculations, and setting conditions.

**Operators** – SQL uses various operators to perform comparisons and logic. Arithmetic operators like +, -, \*, / handle math, comparison operators such as =, >, < compare values, and logical operators like AND, OR, NOT combine conditions for more complex queries.

**Functions** – SQL provides built-in functions to simplify data handling. Aggregate functions like COUNT(), SUM(), AVG() perform calculations on sets of rows, string functions like UPPER(), LOWER() modify text, and date functions like NOW() or DATE\_FORMAT() handle dates and times.

**Wildcards** – When you’re unsure about the exact value you’re searching for, wildcards help. The % symbol represents any number of characters, while \_ represents a single character. For example, WHERE name LIKE 'A%' finds names starting with "A."

**Comments** – SQL supports comments to explain your code without affecting execution. Use -- for single-line comments or /\* \*/ for multi-line comments. Comments are helpful for documenting complex queries or reminders.

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

Ans.

SELECT column1, column2, ...

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column ASC|DESC

LIMIT number;

3. Explain the role of clauses in SQL statements.

Ans.

Clauses in SQL are essential parts of statements that help refine and control the data you retrieve or manipulate. They work alongside commands like SELECT, UPDATE, and DELETE to make queries more specific and powerful.

example, the WHERE clause filters rows based on conditions, like WHERE age > 18, ensuring only matching records appear. The ORDER BY clause sorts the results in ascending or descending order, helping organize the output. GROUP BY groups rows with the same values — useful for summarizing data — and HAVING applies conditions to those grouped results. Finally, LIMIT restricts how many rows the query returns, which is helpful when you only need a subset of data.

LAB EXERCISES:

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

Ans.

SELECT student\_name, age

FROM students;

• Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

Ans.

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.

1. PRIMARY KEY

2. FOREIGN KEY

3. NOT NULL

4. **UNIQUE**

5. **CHECK**

6. DEFAULT

7. AUTO\_INCREMENT

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

Ans.

PRIMARY KEY: A **PRIMARY KEY** is a constraint that uniquely identifies each record in a table. It ensures that no two rows have the same value in the primary key column, and it also prevents NULL values. This guarantees that every row can be uniquely referenced. Each table can have only one primary key, though it can consist of multiple columns combined (called a composite key). For example, a students table might use student\_id as the primary key to ensure each student has a unique identifier.

FOREIGN KEY: A **FOREIGN KEY**, on the other hand, establishes a relationship between two tables. It links a column in one table to the primary key in another table, ensuring data consistency — meaning you can’t insert a value in the foreign key column unless it already exists in the referenced table. This helps maintain referential integrity. For instance, an enrollments table might include student\_id as a foreign key pointing to the students table, ensuring that only existing students can be added to the enrollments.

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

Ans.

The **NOT NULL** constraint ensures that a column cannot have NULL values. It forces users to provide a value when inserting or updating data, preventing incomplete records. For example, if a students table has a student\_name column defined as NOT NULL, every student must have a name — attempts to insert a record without a name will fail. This constraint is crucial for mandatory data fields like usernames, emails, or primary contact numbers.

The **UNIQUE** constraint ensures that all values in a specific column are different from one another. It prevents duplicates while still allowing NULL values (unless combined with NOT NULL). For example, if an employees table has an email column set to UNIQUE, no two employees can have the same email address. This constraint is commonly used for fields like email IDs, phone numbers, or national identification numbers, where each value must remain distinct.

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 TABLE teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE

);

• Lab 2: Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.

Ans.

CREATE TABLE teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE

);

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

age INT CHECK (age > 0),

class VARCHAR(20) NOT NULL,

address VARCHAR(150),

teacher\_id INT

);

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id)

REFERENCES teachers(teacher\_id)

ON DELETE CASCADE;

INSERT INTO teachers (teacher\_id, teacher\_name, subject, email)

VALUES

(1, 'Mr. Sharma', 'Math', 'sharma.math@example.com'),

(2, 'Ms. Verma', 'Science', 'verma.science@example.com');

INSERT INTO students (student\_id, student\_name, age, class, address, teacher\_id)

VALUES

(1, 'Aarav Sharma', 14, '8A', '123 Dindoli', 1),

(2, 'Ishita Verma', 15, '9B', '456 Godadara', 2),

(3, 'Rohan Patel', 13, '7C', '789 Rander', 1);

DELETE FROM teachers WHERE teacher\_id = 1;

SELECT \* FROM students;

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

Theory Questions:

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

Ans.

**SQL Data Definition Language (DDL)** consists of commands that define, modify, and manage the structure of database objects like tables, schemas, indexes, and views. DDL commands don’t manipulate the data itself — instead, they focus on setting up and changing the database’s blueprint. When a DDL command is executed, it’s automatically committed, meaning the changes are saved permanently. This ensures the structure remains consistent.

Key DDL commands include:

**CREATE:** Defines new tables, databases, or other objects.

**ALTER:** Modifies existing structures, like adding a column to a table.

**DROP:** Deletes tables, databases, or objects permanently.

**TRUNCATE:** Removes all data from a table but keeps its structure intact.

**RENAME:** Changes the name of a table or object.

2. Explain the CREATE command and its syntax.

Ans.

The **CREATE** command in SQL is part of **DDL (Data Definition Language)** and is used to create new database objects such as tables, databases, views, or indexes. It defines the structure of the object, including columns, data types, and constraints.

CREATE TABLE table\_name (

column1 data\_type constraint,

column2 data\_type constraint,

...

);

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

Ans.

Purpose of Data Types:

**Defines the kind of data** a column can hold — like INT for numbers, VARCHAR for text, or DATE for dates.

**Saves storage space** by allocating only the necessary memory. For example, TINYINT uses less space than BIGINT.

**Improves performance** by allowing faster retrieval and processing of data when the correct type is used.

**Prevents errors** by rejecting invalid data — like trying to store text in a numeric column.

Purpose of Constraints:

**Enforce data rules** to ensure accuracy and reliability.

**Prevent duplicate data** using constraints like PRIMARY KEY and UNIQUE.

**Ensure mandatory values** with NOT NULL so critical fields can’t be left empty.

**Maintain relationships** between tables using FOREIGN KEY constraints.

**Restrict invalid values** using CHECK (e.g., ensuring age > 0).

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 TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

course\_credits INT CHECK (course\_credits > 0)

);

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

Ans.

CREATE DATABASE university\_db;

5. ALTER Command

Theory Questions:

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

Ans.  
The **ALTER** command in SQL is used to **modify an existing database structure** — mainly tables — without losing the existing data. It allows you to make changes like adding, modifying, or deleting columns and constraints after the table has been created.

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

Ans.

1. Add a New Column

ALTER TABLE students

ADD phone\_number VARCHAR(15);

**2. Modify an Existing Column**

ALTER TABLE students

MODIFY age INT NOT NULL;

3. Drop a Column

ALTER TABLE students

DROP COLUMN phone\_number;

LAB EXERCISES:

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

Ans.

ALTER TABLE courses

ADD course\_duration VARCHAR(20);

• Lab 2: Drop the course\_credits column from the courses table.

Ans.

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 in SQL is used to **permanently** delete database objects like tables, databases, or views. It completely removes the structure and data associated with the object. Once executed, this action **cannot be undone**, so it should be used carefully.

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

Ans.

Dropping a table from a database has several important implications:

**Permanent Data Loss:** All data stored within the table is permanently deleted and cannot be recovered (unless you have backups).

**Structure Removal:** The table's structure, including columns, data types, and constraints, is completely removed from the database.

**Dependency Issues:** If other tables reference the dropped table through **foreign keys**, this can cause errors or break relationships, unless the foreign key constraint is set to **CASCADE** (which also deletes related rows).

**Loss of Permissions and Indexes:** Any associated indexes, triggers, or permissions specific to that table are also lost.

LAB EXERCISES:

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

Ans.

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.

Ans.

USE school\_db;

DROP TABLE students;

SHOW TABLES;

7. Data Manipulation Language (DML)

Theory Questions:

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

Ans.

INSERT Command

INSERT INTO table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

UPDATE Command

UPDATE table\_name

SET column1 = value1, column2 = value2

WHERE condition;

DELETE Command

DELETE FROM table\_name

WHERE condition;

2. What isthe importance of the WHERE clause in UPDATE and DELETE operations?

Ans.

UPDATE Operations:

UPDATE students

SET age = 17

WHERE student\_name = 'Kamlesh';

DELETE Operations:

DELETE FROM students

WHERE student\_name = 'Kamlesh';

LAB EXERCISES:

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

Ans.

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

VALUES

(1, 'Mathematics', 4, '6 months'),

(2, 'Computer Science', 5, '1 year'),

(3, 'Physics', 3, '6 months');

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

Ans.

UPDATE courses

SET course\_duration = '8 months'

WHERE course\_name = 'Mathematics';

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

Ans.

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?

Ans.

The SELECT statement is one of the most important SQL commands — it’s used to **retrieve data** from a database table. It allows you to fetch specific columns, rows, or even apply conditions and sorting to get exactly what you need.

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

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

Ans.

The WHERE clause is used to **filter rows** based on a condition. It ensures only the rows that meet the criteria are included in the result set.

SELECT column1, column2

FROM table\_name

WHERE condition;

LAB EXERCISES:

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

Ans.

SELECT \* FROM courses;

• Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.

Ans.

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.

Ans.

SELECT \* FROM courses

LIMIT 2;

9. Data Control Language (DCL)

Theory Questions:

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

Ans.

The GRANT command gives specific **privileges** (permissions) to users or roles on a database, table, or even specific columns.

GRANT privilege ON table\_name TO user;

The REVOKE command removes previously given permissions from a user or role.

REVOKE privilege ON table\_name FROM user;

2. How do you manage privileges using these commands?

Ans.

1. Granting Privileges

You assign specific permissions to a user using the GRANT command.

GRANT privilege(s) ON database\_name.table\_name TO 'user'@'host';

**2. Checking Privileges**

You can view a user’s current permissions:

SHOW GRANTS FOR 'kamlesh'@'localhost';

Revoking Privileges

If a user no longer needs a permission, you can remove it with REVOKE:

REVOKE privilege(s) ON database\_name.table\_name FROM 'user'@'host';

REVOKE INSERT ON school\_db.students FROM 'kamlesh'@'localhost';

LAB EXERCISES:

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

Ans.

Create Two New Users:

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

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

Grant SELECT Permission to user

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

**Verify Permissions** (Optional)

SHOW GRANTS FOR 'user1'@'localhost';

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

Ans.

Revoke INSERT Permission from user1

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

Grant INSERT Permission to user2

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

**Verify Permissions** (Optional)

SHOW GRANTS FOR 'user1'@'localhost';

SHOW GRANTS FOR 'user2'@'localhost';

10. Transaction Control Language (TCL)

Theory Questions:

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

Ans.

COMMIT Command — Save Changes Permanently

The COMMIT command **saves all the changes** made during the current transaction to the database permanently. Once a COMMIT is executed, the changes **can’t be undone**.

ROLLBACK Command — Undo Changes

The ROLLBACK command **reverses** any changes made during the current transaction if a mistake happens — but only if you haven’t committed yet.

2. Explain how transactions are managed in SQL databases.

Ans.

Managing Transactions in SQL Databases

**transaction** in SQL is a series of one or more SQL operations executed as a **single, logical unit of work**. The goal is to ensure that the database remains **consistent** and **reliable**, even in the face of errors, crashes, or multiple users accessing the data simultaneously.

Transactions follow the **ACID** properties:

* **Atomicity:** The transaction is treated as a single unit — **all** changes happen, or **none** do.  
  Example: Transferring money between accounts — both debit and credit must happen, or neither should.
* **Consistency:** The database must remain valid and follow all rules (like constraints) before and after the transaction.  
  Example: A student’s age can’t become a negative number.
* **Isolation:** Transactions run independently, ensuring **concurrent** transactions don’t interfere with each other.  
  Example: Two users booking the last train ticket won’t both succeed.
* **Durability:** Once a transaction is committed, the changes are **permanent**, even if the system crashes.  
  Example: If a new course is added to the database and the system restarts, the course data remains saved.

LAB EXERCISES:

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

Ans.

BEGIN TRANSACTION;

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

VALUES

(101, 'Mathematics', 4, '6 months'),

(102, 'Physics', 3, '5 months'),

(103, 'Computer Science', 4, '8 months');

COMMIT;

SELECT \* FROM courses;

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

Ans.

BEGIN TRANSACTION;

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

VALUES

(104, 'Chemistry', 3, '6 months'),

(105, 'Biology', 3, '6 months'),

(106, 'History', 2, '4 months');

ROLLBACK;

SELECT \* FROM courses;

• Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans.

BEGIN TRANSACTION;

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

VALUES

(107, 'Economics', 3, '5 months'),

(108, 'Philosophy', 2, '4 months');

SAVEPOINT before\_update;

UPDATE courses SET course\_duration = '7 months' WHERE course\_name = 'Economics';

UPDATE courses SET course\_duration = '3 months' WHERE course\_name = 'Philosophy';

ROLLBACK TO before\_update;

COMMIT;

SELECT \* FROM courses;

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** in SQL is used to **combine data** from two or more tables based on a **related column** (usually a primary key in one table and a foreign key in another).

INNER JOIN:

Returns **only matching** rows from both tables.

LEFT JOIN:

Returns **all rows from the left table** + matching rows from the right table.

RIGHT JOIN:

Returns **all rows from the right table** + matching rows from the left table.

FULL OUTER JOIN:

Returns **all rows from both tables**, filling missing values with NULL.

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

Ans.

Joins in SQL allow us to **combine data** from multiple tables based on a **common column**, usually a **primary key in one table** and a **foreign key in another**. This helps in retrieving meaningful information from a relational database.

LAB EXERCISES:

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

Ans.

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50) NOT NULL

);

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50) NOT NULL,

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

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

VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

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

VALUES

(101, 'Kamlesh Mali', 1),

(102, 'Aarav Patel', 2),

(103, 'Riya Shah', 3),

(104, 'Vikram Singh', 2);

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.

Ans.

A LEFT JOIN returns **all records from the left table (departments)** and **matching records from the right table (employees)**. If no match is found, it fills the missing values with NULL.

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?

Ans.

The GROUP BY clause is used to **group rows** that have the same values in one or more columns. It is commonly used with **aggregate functions** like COUNT(), SUM(), AVG(), MAX(), and MIN() to perform calculations on each group.

SELECT column\_name, AGGREGATE\_FUNCTION(column\_name)

FROM table\_name

GROUP BY column\_name;

SELECT department\_id, COUNT(employee\_id) AS total\_employees

FROM employees

GROUP BY department\_id;

SELECT department\_id, COUNT(employee\_id) AS total\_employees

FROM employees

GROUP BY department\_id

HAVING COUNT(employee\_id) > 2;

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

Ans.

GROUP BY:

Groups rows with similar values

Aggregate functions (COUNT(), SUM(), AVG(), etc.)

Returns fewer rows (one per group)

Comes **before** ORDER BY in queries

ORDER BY:

Sorts rows based on column values

Any column, even without aggregation

Returns all rows in sorted order

Comes **after** GROUP BY

LAB EXERCISES:

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

Ans.

SELECT department\_id, COUNT(employee\_id) AS total\_employees

FROM employees

GROUP BY department\_id;

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

Ans.

SELECT department\_id, AVG(salary) AS average\_salary

FROM employees

GROUP BY department\_id;

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 SQL program** that is **stored in the database** and can be executed whenever needed. It **contains multiple SQL statements** (such as SELECT, INSERT, UPDATE, DELETE) and **accepts parameters** to make queries more dynamic.

Stored Procedure:

Precompiled and stored in the database

Faster due to precompilation

Can be reused multiple times

Can restrict access using permissions

Supports variables, loops, and conditions

Standard SQL Query:

Runs when executed Processed each time it's run

Must be written again each time

Less secure, as queries can be modified

Only executes a single query at a time

2. Explain the advantages of using stored procedures.

Ans.

Stored procedures are **precompiled** and stored in the database.

Users can be granted permission to execute a stored procedure **without** accessing the underlying tables.

Stored procedures **can be used multiple times** across different applications.

When a stored procedure runs, it executes **on the database server**.

Complex business logic can be **organized into stored procedures**.

Since stored procedures **centralize business logic**, they ensure **uniform execution** of operations.

LAB EXERCISES:

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

Ans.

DELIMITER //

CREATE PROCEDURE GetEmployeesByDepartment(IN dept\_id INT)

BEGIN

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

FROM employees

WHERE department\_id = dept\_id;

END //

DELIMITER ;

CALL GetEmployeesByDepartment(2);

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

Ans.

DELIMITER //

CREATE PROCEDURE GetCourseDetails(IN course\_id\_input INT)

BEGIN

SELECT course\_id, course\_name, course\_credits, course\_duration

FROM courses

WHERE course\_id = course\_id\_input;

END //

DELIMITER ;

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** in SQL is a **virtual table** that is based on the result of a **SELECT query**. It does not store data itself but displays data from one or more tables. Views help in simplifying complex queries, securing sensitive data, and improving query readability.

View:

Does **not** store data physically

Based on a **query** from one or more tables

Can be updated (with some limitations)

Table:

Stores data physically in the database

Contains actual records

Can be modified directly

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

Ans.

Views can **restrict access** to specific columns or rows.

Views **store frequently used queries**, reducing the need to write complex joins and filters repeatedly.

Views present data in a **structured format**, hiding underlying table complexities.

In some cases, **indexed views** can speed up query execution.

If multiple reports depend on a complex SQL query, creating a view avoids modifying multiple queries.

Views can be used to **summarize and aggregate** data using functions like SUM(), AVG(), and COUNT().

LAB EXERCISES:

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

Ans.

CREATE VIEW EmployeeDepartmentView AS

SELECT e.employee\_id, e.employee\_name, e.salary, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

SELECT \* FROM EmployeeDepartmentView;

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

Ans.

DROP VIEW IF EXISTS EmployeeDepartmentView;

CREATE VIEW EmployeeDepartmentView AS

SELECT e.employee\_id, e.employee\_name, e.salary, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE e.salary >= 50000;

SELECT \* FROM EmployeeDepartmentView;

15. SQL Triggers

Theory Questions:

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

Ans.

A **trigger** in SQL is a special type of stored procedure that automatically executes when a specific event occurs in a database. Triggers help enforce business rules, maintain data integrity, and automate tasks.

BEFORE Triggers

**executed before** an INSERT, UPDATE, or DELETE operation occurs.  
Used to **validate data** before it is modified in the table.

AFTER Triggers

**Executed after** an INSERT, UPDATE, or DELETE operation.  
Used to **log changes** or **update related tables**.

INSTEAD OF Triggers

Used **on views** to handle modifications when direct changes are not allowed.  
Used to **control updates on complex views** that combine multiple tables.

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

Ans.

INSERT:

A new row is added

Log new records, set default values, enforce validation

UPDATE:

An existing row is modified

Track changes, enforce rules, synchronize data

DELETE:

A row is removed

Prevent deletion, archive data, maintain logs

LAB EXERCISES:

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

Ans.

CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

employee\_name VARCHAR(100),

action VARCHAR(50),

action\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TRIGGER AfterInsertEmployee

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_log (employee\_id, employee\_name, action, action\_date)

VALUES (NEW.employee\_id, NEW.employee\_name, 'INSERT', NOW());

END;

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

VALUES (101, 'John Doe', 55000, 1);

SELECT \* FROM employee\_log;

• Lab 2: Create a trigger to update the last\_modified timestamp whenever an employee record is updated.

Ans.

ALTER TABLE employees

ADD COLUMN last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP;

CREATE TRIGGER BeforeUpdateEmployee

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

SET NEW.last\_modified = NOW();

END;

UPDATE employees

SET salary = 60000

WHERE employee\_id = 101;

SELECT employee\_id, employee\_name, salary, last\_modified

FROM employees

WHERE employee\_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/Structured Query Language**) is an **extension of SQL** developed by **Oracle**. It adds **procedural programming** features such as **variables, loops, conditions, and error handling** to SQL, allowing developers to write **complex business logic** within the database.

Unlike standard SQL, PL/SQL allows the use of **loops (FOR, WHILE)**, **conditions (IF-THEN-ELSE)**, and **exception handling** for better control over logic.

PL/SQL organizes code into **blocks** (anonymous blocks, procedures, functions, triggers, and packages) for modular and reusable programming.

Unlike SQL, which only executes individual queries, PL/SQL enables **reusable** procedures and functions, improving performance.

PL/SQL reduces network overhead by **executing multiple SQL statements as a single block**, minimizing the number of database calls.

SQL alone doesn’t have error handling. PL/SQL includes **EXCEPTION blocks** to handle **runtime errors** gracefully.

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

Ans.

Supports Procedural Programming:

PL/SQL allows the use of **loops (FOR, WHILE)**, **conditional statements (IF-THEN-ELSE)**, and **functions**, making it more powerful than standard SQL.

Reduces Network Traffic:

Since PL/SQL **processes multiple SQL statements in a single block**, it minimizes **database server calls**, improving performance.

Enhances Performance with Caching & Bulk Operations:

PL/SQL supports **bulk processing (BULK COLLECT, FORALL)**, making it faster when handling large datasets.

Improves Code Reusability & Modularity:

PL/SQL allows developers to create **stored procedures, functions, and packages**, promoting reusable and structured code.

Provides Robust Error Handling:

PL/SQL includes **exception handling** (EXCEPTION block), allowing smooth execution and preventing failures.

Ensures Data Security:

PL/SQL allows developers to control **who can access and modify data** by using **stored procedures and functions** instead of direct table access.

Allows Transaction Management:

PL/SQL supports **COMMIT, ROLLBACK, and SAVEPOINT**, allowing developers to manage database transactions effectively.

Benefit:

Procedural Capabilities

Reduces Network Traffic

Enhances Performance

Code Reusability

**Error Handling**

Data Security

Transaction Control

LAB EXERCISES:

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

Ans.

DECLARE

v\_total\_employees NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_total\_employees FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Total number of employees: ' || v\_total\_employees);

END;

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

Ans.

DECLARE

v\_total\_sales NUMBER;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Total Sales: ' || v\_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 developers to control the flow of execution based on conditions and iterations. They are categorized into **conditional statements** (e.g., IF-THEN) and **looping constructs** (e.g., LOOP, FOR, WHILE).

The IF-THEN statement is used to execute a block of code only if a specified condition is met.

IF condition THEN

-- Statements to execute if the condition is TRUE

END IF;

LOOP Control Structures:

(a) Simple LOOP:

Executes repeatedly until an explicit EXIT statement is encountered.

LOOP

-- Statements

EXIT WHEN condition;

END LOOP;

(b) FOR LOOP:

Executes a fixed number of times.

FOR counter\_variable IN start\_value..end\_value LOOP

-- Statements

END LOOP;

(c) WHILE LOOP:

Repeats as long as a condition is TRUE.

WHILE condition LOOP

-- Statements

END LOOP;

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

Ans.

Control structures in PL/SQL enhance the flexibility and efficiency of queries by allowing **conditional logic, loops, and decision-making**, making them suitable for handling **dynamic and complex data operations**.

Conditional Execution (IF-THEN):

Dynamically changes query logic based on conditions.

Loops (LOOP, FOR, WHILE):

Automates bulk data processing and retrieval.

Cursor Loops:

Efficiently processes query results row by row.

Exception Handling:

Prevents errors and ensures smooth execution.

LAB EXERCISES:

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

Ans.

DECLARE

v\_emp\_id employees.employee\_id%TYPE := 101;

v\_dept\_name departments.department\_name%TYPE;

BEGIN

SELECT d.department\_name

INTO v\_dept\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE e.employee\_id = v\_emp\_id;

IF v\_dept\_name = 'IT' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ' || v\_emp\_id || ' works in the IT department.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee ' || v\_emp\_id || ' works in the ' || v\_dept\_name || ' department.');

END IF;

END;

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

Ans.

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_name FROM employees;

BEGIN

FOR emp\_record IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('EmployeeName:'||emp\_record.employee\_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** in PL/SQL is a pointer or control structure used to retrieve and process multiple rows returned by a query, one at a time. Cursors help in handling result sets efficiently, allowing row-by-row operations.

Implicit Cursor:

Automatically created by PL/SQL when executing SELECT INTO, INSERT, UPDATE, or DELETE statements.

PL/SQL manages opening, fetching, and closing automatically.

Best for single-row queries.

More efficient for simple operations.

Explicit Cursor:

Manually defined by the programmer using the CURSOR keyword for queries returning multiple rows.

The programmer must explicitly declare, open, fetch, and close the cursor.

Used when handling multiple rows.

Provides better control for complex queries and multiple records.

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

Ans.

**Handling Multiple Rows** – When a query returns multiple rows, an explicit cursor allows you to fetch and process each row individually.

**Better Control Over Data Processing** – Explicit cursors allow you to control when to open, fetch, and close the cursor, making them useful for complex logic.

**Avoiding Exceptions in Multi-Row Queries** – Implicit cursors can cause the TOO\_MANY\_ROWS exception if a SELECT INTO statement returns more than one row. Explicit cursors handle such cases gracefully.

**Looping Through Result Sets** – When you need to iterate over a result set using a **LOOP, WHILE, or FOR LOOP**, explicit cursors provide flexibility.

**Reducing Memory Usage** – Explicit cursors help manage large datasets efficiently by fetching only one row at a time instead of loading all data into memory at once.

LAB EXERCISES:

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

Ans.

DECLARE

CURSOR emp\_cursor IS

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

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

v\_emp\_name employees.employee\_name%TYPE;

v\_dept\_id employees.department\_id%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_emp\_id, v\_emp\_name, v\_dept\_id, v\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_emp\_id || ', Name: ' || v\_emp\_name ||

', Dept ID: ' || v\_dept\_id || ', Salary: ' || v\_salary);

END LOOP;

CLOSE emp\_cursor;

END;

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

Ans.

DECLARE

CURSOR course\_cursor IS

SELECT course\_id, course\_name, course\_duration FROM courses;

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

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

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

BEGIN

OPEN course\_cursor;

LOOP

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

EXIT WHEN course\_cursor%NOTFOUND;

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

', Duration: ' || v\_course\_duration);

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 save points?

Ans.

A **SAVEPOINT** in SQL is used to set a checkpoint within a transaction. It allows partial rollbacks, meaning you can undo only specific parts of a transaction instead of rolling back the entire transaction. This is useful when dealing with multiple operations in a transaction and needing to revert only certain steps while keeping others intact.

**COMMIT** – When a COMMIT is issued, all changes made in the transaction (including those before and after the SAVEPOINT) are permanently saved, and all savepoints are removed.

**ROLLBACK TO SAVEPOINT** – Instead of rolling back the entire transaction, you can use ROLLBACK TO savepoint\_name to undo only the changes made after that savepoint while keeping the previous operations intact.

**ROLLBACK (Without SAVEPOINT)** – If ROLLBACK is issued without specifying a savepoint, the entire transaction is undone, discarding all changes.

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

Ans.

**SAVEPOINTs** are useful in scenarios where you need finer control over transaction rollbacks. Instead of undoing the entire transaction, you can revert only specific operations while keeping others intact. Here are some key situations where using SAVEPOINTs is beneficial:

1. Handling Partial Rollbacks in Complex Transactions:

When a transaction consists of multiple SQL operations, and only a part of it needs to be undone while keeping the rest intact.

2. Avoiding Full Transaction Rollbacks:

If an error occurs in one part of the transaction, instead of rolling back everything, you can revert only the problematic changes.

3. Nested Transactions:

In applications where sub-transactions need to be handled separately within a larger transaction, SAVEPOINTs allow selective rollbacks without affecting the entire process.

4. Debugging and Testing Transactions:

During development, SAVEPOINTs help in debugging by allowing partial rollbacks and testing different parts of a transaction.

5. Error Recovery Without Aborting the Whole Transaction:

If a specific step in a transaction fails, you can revert only that part and proceed with the remaining operations.

LAB EXERCISES:

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

Ans.

START TRANSACTION;

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

VALUES (201, 'SQL Basics', '3 Months');

SAVEPOINT sp1;

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

VALUES (202, 'PL/SQL Advanced', '4 Months');

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

VALUES (203, 'Database Security', '2 Months');

ROLLBACK TO sp1;

COMMIT;

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

Ans.

START TRANSACTION;

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

VALUES (301, 'Data Structures', '5 Months');

SAVEPOINT sp1;

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

VALUES (302, 'Machine Learning', '6 Months');

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

VALUES (303, 'Cloud Computing', '4 Months');

COMMIT;

ROLLBACK TO sp1;

COMMIT;

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.

Ans.

CREATE DATABASE library\_db;

USE library\_db;

CREATE TABLE books (

book\_id INT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(255) NOT NULL,

publisher VARCHAR(255),

year\_of\_publication INT,

price DECIMAL(10,2)

);

INSERT INTO books (book\_id, title, author, publisher, year\_of\_publication, price) VALUES

(1, 'The Discovery of India', 'Jawaharlal Nehru', 'The Signet Press', 1946, 399.00),

(2, 'Wings of Fire', 'A.P.J. Abdul Kalam', 'Universities Press', 1999, 250.00),

(3, 'God of Small Things', 'Arundhati Roy', 'IndiaInk', 1997, 350.00),

(4, 'Train to Pakistan', 'Khushwant Singh', 'Ravi Dayal Publisher', 1956, 299.00),

(5, 'The White Tiger', 'Aravind Adiga', 'HarperCollins India', 2008, 450.00);

• 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

Ans.

USE library\_db;

CREATE TABLE members (

member\_id INT PRIMARY KEY,

member\_name VARCHAR(255) NOT NULL,

date\_of\_membership DATE NOT NULL,

email VARCHAR(255) UNIQUE

);

INSERT INTO members (member\_id, member\_name, date\_of\_membership, email) VALUES

(1, 'Rajesh Kumar', '2022-01-15', 'rajesh.kumar@gmail.com'),

(2, 'Priya Sharma', '2021-11-20', 'priya.sharma@gmail.com'),

(3, 'Amit Verma', '2023-03-10', 'amit.verma@gmail.com'),

(4, 'Neha Singh', '2022-07-05', 'neha.singh@gmail.com'),

(5, 'Suresh Patil', '2021-09-30', 'suresh.patil@gmail.com');

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.

Ans.

SELECT \* FROM members

WHERE date\_of\_membership < '2022-01-01'

ORDER BY date\_of\_membership ASC;

• 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.

Ans.

SELECT title

FROM books

WHERE author = 'Arundhati Roy'

ORDER BY year\_of\_publication DESC;

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.

Ans.

ALTER TABLE books

ADD CONSTRAINT chk\_price CHECK (price > 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.

Ans.

ALTER TABLE members

ADD CONSTRAINT unique\_email UNIQUE (email);

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.

Ans.

CREATE TABLE authors (

author\_id INT PRIMARY KEY,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

country VARCHAR(100) NOT NULL

);

• 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.

Ans.

CREATE TABLE publishers (

publisher\_id INT PRIMARY KEY,

publisher\_name VARCHAR(255) NOT NULL,

contact\_number VARCHAR(20) UNIQUE NOT NULL,

address VARCHAR(255) NOT NULL

);

5. ALTER Command

LAB EXERCISES:

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

Ans.

ALTER TABLE books

ADD genre VARCHAR(100) NOT NULL;

UPDATE books

SET genre = 'Fiction'

WHERE title = 'The God of Small Things';

UPDATE books

SET genre = 'Mythology'

WHERE title = 'The Immortals of Meluha';

UPDATE books

SET genre = 'Historical Fiction'

WHERE title = 'Train to Pakistan';

UPDATE books

SET genre = 'Science Fiction'

WHERE title = 'The Three-Body Problem';

UPDATE books

SET genre = 'Self-Help'

WHERE title = 'The Power of Your Subconscious Mind';

• Lab 4: Modify the members table to increase the length of the email column to 100 characters.

Ans.

ALTER TABLE members

MODIFY email VARCHAR(100);

6. DROP Command

LAB EXERCISES:

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

Ans.

DESC publishers;

SHOW CREATE TABLE publishers;

DROP TABLE publishers;

• Lab 4: Create a backup of the members table and then drop the original members table.

Ans.

CREATE TABLE members\_backup AS

SELECT \* FROM members;

DROP TABLE members;

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.

Ans.

INSERT INTO authors (author\_id, first\_name, last\_name, country)

VALUES

(1, 'Chetan', 'Bhagat', 'India'),

(2, 'Amish', 'Tripathi', 'India'),

(3, 'Ruskin', 'Bond', 'India');

UPDATE authors

SET last\_name = 'Sharma'

WHERE author\_id = 2;

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

Ans.

DELETE FROM books

WHERE price > 100;

8. UPDATE Command

LAB EXERCISES:

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

Ans.

UPDATE books

SET year\_of\_publication = 2023

WHERE book\_id = 2;

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

Ans.

UPDATE books

SET price = price \* 1.10

WHERE year\_of\_publication < 2015;

9. DELETE Command

LAB EXERCISES:

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

Ans.

DELETE FROM members

WHERE date\_of\_membership < '2020-01-01';

SELECT \* FROM members

WHERE date\_of\_membership < '2020-01-01';

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

Ans.

DELETE FROM books

WHERE author IS NULL;

SELECT \* FROM books

WHERE author IS NULL;

10. Data Query Language (DQL)

LAB EXERCISES:

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

Ans.

SELECT \* FROM books

WHERE 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.

Ans.

SELECT \* FROM books

ORDER BY author ASC

LIMIT 3;

11. Data Control Language (DCL)

LAB EXERCISES:

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

Ans.

GRANT SELECT ON books TO librarian;

SHOW GRANTS FOR librarian;

• Lab 4: Grant INSERT and UPDATE permissions to the user admin on the members table.

Ans.

GRANT INSERT, UPDATE ON members TO admin;

SHOW GRANTS FOR admin;

12. REVOKE Command

LAB EXERCISES:

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

Ans.

REVOKE INSERT ON books FROM librarian;

SHOW GRANTS FOR librarian;

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

Ans.

REVOKE ALL PRIVILEGES ON members FROM admin;

SHOW GRANTS FOR admin;

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.

Ans.

START TRANSACTION;

INSERT INTO books (book\_id, title, author, publisher, year\_of\_publication, price, genre)

VALUES

(101, 'The Guide', 'R. K. Narayan', 'Indian Thought Publications', 1958, 350, 'Fiction'),

(102, 'Malgudi Days', 'R. K. Narayan', 'Indian Thought Publications', 1943, 299, 'Short Stories'),

(103, 'The White Tiger', 'Aravind Adiga', 'HarperCollins India', 2008, 450, 'Fiction');

COMMIT;

START TRANSACTION;

INSERT INTO books (book\_id, title, author, publisher, year\_of\_publication, price, genre)

VALUES (104, 'Midnight’s Children', 'Salman Rushdie', 'Vintage Books', 1981, 500, 'Historical Fiction');

ROLLBACK;

SELECT \* FROM books;

• Lab 4: Set a SAVEPOINT before making updates to the members table, perform some updates, and then roll back to the SAVEPOINT.

Ans.

START TRANSACTION;

SAVEPOINT before\_update;

UPDATE members

SET email = 'updated\_email1@Agmail.com'

WHERE member\_id = 1;

UPDATE members

SET email = 'updated\_email2@Agmail.com'

WHERE member\_id = 2;

ROLLBACK TO before\_update;

COMMIT;

SELECT \* FROM members;

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.

Ans.

SELECT books.title, authors.first\_name, authors.last\_name

FROM books

INNER JOIN authors ON books.author\_id = authors.author\_id;

• 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.

Ans.

SELECT books.title, authors.first\_name, authors.last\_name

FROM books

FULL OUTER JOIN authors ON books.author\_id = authors.author\_id;

SELECT books.title, authors.first\_name, authors.last\_name

FROM books

LEFT JOIN authors ON books.author\_id = authors.author\_id

UNION

SELECT books.title, authors.first\_name, authors.last\_name

FROM books

RIGHT JOIN authors ON books.author\_id = authors.author\_id;

15. SQL Group By

LAB EXERCISES:

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

Ans.

SELECT genre, COUNT(\*) AS total\_books

FROM books

GROUP BY genre;

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

Ans.

SELECT YEAR(date\_of\_membership) AS join\_year, COUNT(\*) AS total\_members

FROM members

GROUP BY YEAR(date\_of\_membership)

ORDER BY join\_year;

16. SQL Stored Procedure

LAB EXERCISES:

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

Ans.

DELIMITER $$

CREATE PROCEDURE GetBooksByAuthor(IN author\_name VARCHAR(255))

BEGIN

SELECT title, year\_of\_publication, genre, price

FROM books

WHERE author = author\_name

ORDER BY year\_of\_publication DESC;

END $$

DELIMITER ;

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

Ans.

DELIMITER $$

CREATE PROCEDURE GetBookPrice(IN book\_id\_param INT, OUT book\_price DECIMAL(10,2))

BEGIN

SELECT price INTO book\_price

FROM books

WHERE book\_id = book\_id\_param;

END $$

DELIMITER ;

CALL GetBookPrice(101, @price);

SELECT @price AS Book\_Price;

17. SQL View

LAB EXERCISES:

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

Ans.

CREATE VIEW BookDetails AS

SELECT title, author, price

FROM books;

SELECT \* FROM BookDetails;

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

Ans.

CREATE VIEW EarlyMembers AS

SELECT \*

FROM members

WHERE YEAR(date\_of\_membership) < 2020;

SELECT \* FROM EarlyMembers;

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.

Ans.

CREATE TRIGGER update\_books\_timestamp

BEFORE UPDATE ON books

FOR EACH ROW

BEGIN

SET NEW.last\_modified = NOW();

END;

• 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.

Ans.

CREATE TABLE log\_changes (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

book\_id INT,

title VARCHAR(255),

deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

action VARCHAR(10)

);

CREATE TRIGGER log\_book\_deletion

AFTER DELETE ON books

FOR EACH ROW

BEGIN

INSERT INTO log\_changes (book\_id, title, action)

VALUES (OLD.book\_id, OLD.title, 'DELETE');

END;

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.

Ans.

DECLARE

v\_book\_id INT := 101;

v\_title VARCHAR2(255) := 'The Guide';

v\_author VARCHAR2(255) := 'R. K. Narayan';

v\_publisher VARCHAR2(255) := 'Indian Thought Publications';

v\_year\_of\_publication INT := 1958;

v\_price NUMBER := 299.50;

BEGIN

INSERT INTO books (book\_id, title, author, publisher, year\_of\_publication, price)

VALUES (v\_book\_id, v\_title, v\_author, v\_publisher, v\_year\_of\_publication, v\_price);

DBMS\_OUTPUT.PUT\_LINE('Book "' || v\_title || '" has been added successfully.');

END;

/

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

Ans.

DECLARE

v\_total\_books NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_total\_books FROM books;

DBMS\_OUTPUT.PUT\_LINE('Total number of books: ' || v\_total\_books);

END;

/

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.

Ans.

DECLARE

v\_book\_id NUMBER := 101;

v\_price NUMBER := 250.50;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || v\_book\_id);

DBMS\_OUTPUT.PUT\_LINE('Price: $' || v\_price);

END;

/

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

Ans.

DECLARE

c\_discount CONSTANT NUMBER := 0.10;

c\_tax CONSTANT NUMBER := 0.05;

v\_original\_price NUMBER := 500;

v\_discounted\_price NUMBER;

v\_final\_price NUMBER;

BEGIN

v\_discounted\_price := v\_original\_price - (v\_original\_price \* c\_discount);

v\_final\_price := v\_discounted\_price + (v\_discounted\_price \* c\_tax);

DBMS\_OUTPUT.PUT\_LINE('Original Price: $' || v\_original\_price);

DBMS\_OUTPUT.PUT\_LINE('Discounted Price: $' || v\_discounted\_price);

DBMS\_OUTPUT.PUT\_LINE('Final Price after Tax: $' || v\_final\_price);

END;

/

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.

Ans.

DECLARE

v\_book\_price NUMBER := 120;

BEGIN

IF v\_book\_price > 100 THEN

DBMS\_OUTPUT.PUT\_LINE('The book is expensive.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('The book is affordable.');

END IF;

END;

/

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

Ans.

DECLARE

CURSOR book\_cursor IS

SELECT book\_id, title, author, price FROM books;

v\_book\_id books.book\_id%TYPE;

v\_title books.title%TYPE;

v\_author books.author%TYPE;

v\_price books.price%TYPE;

BEGIN

FOR book\_rec IN book\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || book\_rec.book\_id ||

', Title: ' || book\_rec.title ||

', Author: ' || book\_rec.author ||

', Price: ' || book\_rec.price);

END LOOP;

END;

/

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.

Ans.

DECLARE

CURSOR member\_cursor IS

SELECT member\_id, member\_name, date\_of\_membership, email FROM members;

v\_member\_id members.member\_id%TYPE;

v\_member\_name members.member\_name%TYPE;

v\_date\_of\_membership members.date\_of\_membership%TYPE;

v\_email members.email%TYPE;

BEGIN

OPEN member\_cursor;

LOOP

FETCH member\_cursor INTO v\_member\_id, v\_member\_name, v\_date\_of\_membership, v\_email;

EXIT WHEN member\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Member ID: ' || v\_member\_id ||

', Name: ' || v\_member\_name ||

', Joined: ' || v\_date\_of\_membership ||

', Email: ' || v\_email);

END LOOP;

CLOSE member\_cursor;

END;

/

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

Ans.

DECLARE

v\_author\_name VARCHAR2(100) := 'Author Name'; -- Replace with the desired author's name

CURSOR book\_cursor IS

SELECT title FROM books WHERE author = v\_author\_name;

v\_title books.title%TYPE;

BEGIN

OPEN book\_cursor;

LOOP

FETCH book\_cursor INTO v\_title;

EXIT WHEN book\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Book Title: ' || v\_title);

END LOOP;

CLOSE book\_cursor;

END;

/

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.

Ans.

START TRANSACTION;

INSERT INTO members (member\_id, member\_name, date\_of\_membership, email)

VALUES (101, 'Amit Sharma', '2024-03-25', 'amitsharma@gmail.com');

SAVEPOINT before\_update;

UPDATE members SET member\_name = 'Amit S.' WHERE member\_id = 101;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;

• 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.

Ans.

START TRANSACTION;

INSERT INTO books (book\_id, title, author, publisher, year\_of\_publication, price) VALUES

(201, 'The Guide', 'R. K. Narayan', 'Indian Thought Publications', 1958, 350),

(202, 'Gitanjali', 'Rabindranath Tagore', 'Macmillan Publishers', 1910, 299),

(203, 'Midnight’s Children', 'Salman Rushdie', 'Jonathan Cape', 1981, 450);

COMMIT;

SAVEPOINT before\_update;

UPDATE books SET price = price + 50 WHERE year\_of\_publication < 2000;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;