**Introduction to DBMS & SQL**

**1. What is SQL, and why is it essential in database management?**  
**Answer:** SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It is essential because it allows users to perform operations such as querying, updating, inserting, and deleting data, as well as defining and controlling database structure and access.

**2. Explain the difference between DBMS and RDBMS.**  
**Answer:** A DBMS (Database Management System) manages data without enforcing relationships among tables. An RDBMS (Relational Database Management System) manages data with enforced relationships, typically using primary and foreign keys, and follows the relational model introduced by E.F. Codd.

**3. Describe the role of SQL in managing relational databases.**  
**Answer:** SQL provides the tools to define, manipulate, and query relational databases. It allows users to create tables, insert and retrieve data, update records, enforce constraints, and control access and transactions.

**4. What are the key features of SQL?**  
**Answer:** Key features include:

* Data Querying
* Data Manipulation
* Data Definition
* Data Control
* Transaction Control
* Use of constraints and joins
* Standardized language for relational databases

**SQL Syntax**

**1. What are the basic components of SQL syntax?**  
**Answer:** Basic components include:

* **Keywords** (e.g., SELECT, INSERT)
* **Identifiers** (e.g., table/column names)
* **Expressions** (e.g., arithmetic or logical)
* **Clauses** (e.g., WHERE, ORDER BY)
* **Operators** (e.g., =, <, >)
* **Comments**

**2. Write the general structure of an SQL SELECT statement.**  
**Answer:**

SELECT column1, column2

FROM table\_name

WHERE condition

ORDER BY column;

**3. Explain the role of clauses in SQL statements.**  
**Answer:** Clauses define the conditions and scope of an SQL query, such as filtering rows (WHERE), sorting (ORDER BY), grouping (GROUP BY), and limiting results (LIMIT).

**SQL Constraints**

**1. What are constraints in SQL? List and explain the different types of constraints.**  
**Answer:** Constraints enforce rules on table data. Types include:

* **NOT NULL**: Ensures a column cannot have NULL values.
* **UNIQUE**: Ensures all values in a column are unique.
* **PRIMARY KEY**: Uniquely identifies each row.
* **FOREIGN KEY**: Ensures referential integrity between tables.
* **CHECK**: Validates data based on a condition.
* **DEFAULT**: Sets a default value if none is provided.

**2. How do PRIMARY KEY and FOREIGN KEY constraints differ?  
Answer:** A PRIMARY KEY uniquely identifies rows within its table. A FOREIGN KEY creates a relationship between two tables and ensures the value exists in the referenced table.

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

* **NOT NULL** prevents a column from accepting NULL values.
* **UNIQUE** ensures all values in a column are distinct.

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

**1. Define the SQL Data Definition Language (DDL).**  
**Answer:** DDL is a subset of SQL used to define and modify database structures like tables, schemas, and indexes using commands such as CREATE, ALTER, and DROP.

**2. Explain the CREATE command and its syntax.**  
**Answer:** CREATE is used to create a database or table.  
Syntax:

sql

CREATE TABLE table\_name (

column1 datatype constraint,

column2 datatype constraint

);

**3. What is the purpose of specifying data types and constraints during table creation?**  
**Answer:** It ensures data integrity, storage optimization, and enforces rules on what type of data can be stored in each column.

**ALTER Command**

**1. What is the use of the ALTER command in SQL?**  
**Answer:** ALTER modifies the structure of an existing table, such as adding, modifying, or deleting columns or constraints.

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

* Add: ALTER TABLE table\_name ADD column\_name datatype;
* Modify: ALTER TABLE table\_name MODIFY column\_name new\_datatype;
* Drop: ALTER TABLE table\_name DROP COLUMN column\_name;

**DROP Command**

**1. What is the function of the DROP command in SQL?**  
**Answer:** DROP removes database objects like tables, views, or databases permanently.

**2. What are the implications of dropping a table from a database?**  
**Answer:** All data and structure of the table are permanently deleted, and it cannot be recovered unless a backup exists.

**Data Manipulation Language (DML)**

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

* **INSERT** adds new rows to a table.
* **UPDATE** modifies existing data.
* **DELETE** removes rows from a table.

**2. What is the importance of the WHERE clause in UPDATE and DELETE operations?**  
**Answer:** It specifies which rows should be updated or deleted. Without WHERE, all rows are affected.

**Data Query Language (DQL)**

**1. What is the SELECT statement, and how is it used to query data?  
Answer:** SELECT retrieves data from tables. It allows filtering, sorting, and aggregation of results.

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

* **WHERE** filters rows based on conditions.
* **ORDER BY** sorts the result set in ascending or descending order.

**Data Control Language (DCL)**

**1. What is the purpose of GRANT and REVOKE in SQL?**  
**Answer:** They manage user permissions. GRANT gives access; REVOKE removes it.

**2. How do you manage privileges using these commands?**  
**Answer:** Use GRANT to assign specific rights (e.g., SELECT, INSERT) to users and REVOKE to take them away.

**Transaction Control Language (TCL)**

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

* **COMMIT** saves all changes made in a transaction.
* **ROLLBACK** undoes changes since the last commit.

**2. Explain how transactions are managed in SQL databases.**  
**Answer:** Transactions group one or more SQL statements into a single unit of work. They follow the ACID properties to ensure data integrity.

**SQL Joins**

**1. Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?**  
**Answer:** Joins combine rows from two or more tables based on related columns.

* **INNER JOIN**: Matches rows in both tables.
* **LEFT JOIN**: All rows from the left table and matched rows from the right.
* **RIGHT JOIN**: All rows from the right table and matched from the left.
* **FULL OUTER JOIN**: All rows when there is a match in either table.

**2. How are joins used to combine data from multiple tables?**  
**Answer:** By using a common key, joins allow combining data from different tables into a single result set.

**SQL GROUP BY**

**1. What is the GROUP BY clause in SQL? How is it used with aggregate functions?**  
**Answer:** GROUP BY groups rows with the same values and allows aggregate functions (like COUNT, SUM) to be applied to each group.

**2. Explain the difference between GROUP BY and ORDER BY.**  
**Answer:**

* GROUP BY groups records based on column values.
* ORDER BY sorts the result set.

**SQL Stored Procedure**

**1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?**  
**Answer:** A stored procedure is a saved SQL code block that performs operations. Unlike a single query, it can include logic, loops, and multiple SQL statements.

**2. Explain the advantages of using stored procedures.**  
**Answer:**

* Code reuse
* Improved performance
* Better security
* Centralized logic

**SQL View**

**1. What is a view in SQL, and how is it different from a table?**  
**Answer:** A view is a virtual table created from a SQL query. Unlike a table, it does not store data physically.

**2. Explain the advantages of using views in SQL databases.**  
**Answer:**

* Simplify complex queries
* Enhance security
* Provide data abstraction
* Present different data perspectives

**SQL Triggers**

**1. What is a trigger in SQL? Describe its types and when they are used.**  
**Answer:** A trigger is a stored procedure that executes automatically in response to certain events (INSERT, UPDATE, DELETE). Types:

* **BEFORE** Trigger
* **AFTER** Trigger
* **INSTEAD OF** Trigger

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

* **INSERT Trigger** activates on new row insertions.
* **UPDATE Trigger** fires when data is modified.
* **DELETE Trigger** responds to data deletion.

**Introduction to PL/SQL**

**1. What is PL/SQL, and how does it extend SQL's capabilities?**  
**Answer:** PL/SQL (Procedural Language/SQL) extends SQL with procedural features like variables, loops, conditions, and functions, enabling more powerful programs.

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

* Supports procedural logic
* Code modularity and reuse
* Enhances performance
* Integrates seamlessly with SQL

**PL/SQL Control Structures**

**1. What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.**  
**Answer:** Control structures manage the flow of execution.

* **IF-THEN** executes code if a condition is true.
* **LOOP** repeats a set of statements until a condition is met.

**2. How do control structures in PL/SQL help in writing complex queries?**  
**Answer:** They enable conditional logic and iteration, making it easier to handle complex business logic in a structured way.

**SQL Cursors**

**1. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.  
Answer:** A cursor is a pointer to a result set.

* **Implicit**: Automatically created for SQL statements.
* **Explicit**: Declared by the user for custom control over result sets.

**2. When would you use an explicit cursor over an implicit one?**  
**Answer:** When you need to process each row individually or perform complex row-by-row operations.

**Rollback and Commit Savepoint**

**1. Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?**  
**Answer:** A SAVEPOINT marks a point in a transaction. ROLLBACK can undo operations to a specific SAVEPOINT, and COMMIT makes all changes permanent.

**2. When is it useful to use savepoints in a database transaction?**  
**Answer:** When you want to partially rollback a transaction without losing all changes—useful for error handling and complex operations.

### ****1. Database and Table Creation****

**Lab 1:**

CREATE DATABASE school\_db;

USE school\_db;

CREATE TABLE students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100),

age INT,

class VARCHAR(20),

address VARCHAR(255)

);

**Lab 2:**

INSERT INTO students VALUES

(1, 'Alice', 12, '6A', '123 Oak St'),

(2, 'Bob', 11, '5B', '456 Maple St'),

(3, 'Charlie', 10, '5A', '789 Pine St'),

(4, 'David', 13, '7B', '321 Cedar St'),

(5, 'Eve', 14, '8A', '654 Birch St');

SELECT \* FROM students;

### ****2. SQL Syntax****

**Lab 1:**

SELECT student\_name, age FROM students;

**Lab 2:**

SELECT \* FROM students WHERE age > 10;

### ****3. SQL Constraints****

**Lab 1:**

CREATE TABLE teachers (

teacher\_id INT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE

);

**Lab 2:**

ALTER TABLE students

ADD teacher\_id INT,

ADD CONSTRAINT fk\_teacher

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

### ****4. DDL Commands****

**Lab 1:**

CREATE TABLE courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(100),

course\_credits INT

);

**Lab 2:**

CREATE DATABASE university\_db;

### ****5. ALTER Command****

**Lab 1:**

ALTER TABLE courses ADD course\_duration INT;

**Lab 2:**

ALTER TABLE courses DROP COLUMN course\_credits;

### ****6. DROP Command****

**Lab 1:**

DROP TABLE teachers;

**Lab 2:**

DROP TABLE students;

### ****7. DML Commands****

**Lab 1:**

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

(101, 'Math', 6),

(102, 'Science', 5),

(103, 'History', 4);

**Lab 2:**

UPDATE courses SET course\_duration = 8 WHERE course\_id = 101;

**Lab 3:**

DELETE FROM courses WHERE course\_id = 103;

### ****8. DQL Commands****

**Lab 1:**

SELECT \* FROM courses;

**Lab 2:**

SELECT \* FROM courses ORDER BY course\_duration DESC;

**Lab 3:**

SELECT \* FROM courses LIMIT 2;

### ****9. DCL Commands****

**Lab 1:**

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

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

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

**Lab 2:**

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

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

### ****10. TCL Commands****

**Lab 1:**

START TRANSACTION;

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

(104, 'English', 6),

(105, 'Art', 3);

COMMIT;

**Lab 2:**

START TRANSACTION;

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

(106, 'Music', 4);

ROLLBACK;

**Lab 3:**

START TRANSACTION;

SAVEPOINT before\_update;

UPDATE courses SET course\_duration = 7 WHERE course\_id = 104;

ROLLBACK TO before\_update;

COMMIT;

### ****11. SQL Joins****

**Lab 1:**

CREATE TABLE departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(100)

);

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(100),

dept\_id INT,

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

);

SELECT e.emp\_name, d.dept\_name

FROM employees e

INNER JOIN departments d ON e.dept\_id = d.dept\_id;

**Lab 2:**

SELECT d.dept\_name, e.emp\_name

FROM departments d

LEFT JOIN employees e ON d.dept\_id = e.dept\_id;

### ****12. SQL GROUP BY****

**Lab 1:**

SELECT dept\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY dept\_id;

**Lab 2:**

SELECT dept\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY dept\_id;

### ****13. SQL Stored Procedure****

**Lab 1:**

CREATE PROCEDURE GetEmployeesByDept(IN dept\_id INT)

BEGIN

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

END;

**Lab 2:**

CREATE PROCEDURE GetCourseDetails(IN cid INT)

BEGIN

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

END;

### ****14. SQL View****

**Lab 1:**

CREATE VIEW emp\_with\_dept AS

SELECT e.\*, d.department\_name

FROM employees e

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

**Lab 2:**

CREATE OR REPLACE VIEW emp\_with\_dept AS

SELECT e.\*, d.department\_name

FROM employees e

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

WHERE e.salary >= 50000;

### ****15. SQL Triggers****

**Lab 1:**

CREATE TRIGGER log\_new\_employee

AFTER INSERT ON employees

FOR EACH ROW

INSERT INTO employee\_log (employee\_id, action, action\_time)

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

**Lab 2:**

CREATE TRIGGER update\_last\_modified

BEFORE UPDATE ON employees

FOR EACH ROW

SET NEW.last\_modified = NOW();

### ****16. Introduction to PL/SQL****

**Lab 1:**

DECLARE

emp\_count INT;

BEGIN

SELECT COUNT(\*) INTO emp\_count FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Total Employees: ' || emp\_count);

END;

**Lab 2:**

DECLARE

total\_sales NUMBER;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Total Sales: ' || total\_sales);

END;

### ****17. PL/SQL Control Structures****

**Lab 1:**

DECLARE

dept\_name VARCHAR2(50);

BEGIN

SELECT department\_name INTO dept\_name FROM departments WHERE department\_id = 10;

IF dept\_name = 'Sales' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee belongs to Sales department.');

END IF;

END;

**Lab 2:**

DECLARE

emp\_name employees.name%TYPE;

BEGIN

FOR emp IN (SELECT name FROM employees) LOOP

DBMS\_OUTPUT.PUT\_LINE(emp.name);

END LOOP;

END;

### ****18. SQL Cursors****

**Lab 1:**

DECLARE

CURSOR emp\_cursor IS SELECT \* FROM employees;

emp\_row employees%ROWTYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO emp\_row;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(emp\_row.name);

END LOOP;

CLOSE emp\_cursor;

END;

**Lab 2:**

DECLARE

CURSOR course\_cursor IS SELECT \* FROM courses;

course\_rec courses%ROWTYPE;

BEGIN

OPEN course\_cursor;

LOOP

FETCH course\_cursor INTO course\_rec;

EXIT WHEN course\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(course\_rec.course\_name);

END LOOP;

CLOSE course\_cursor;

END;

### ****19. Rollback and Commit Savepoint****

**Lab 1:**

START TRANSACTION;

SAVEPOINT sp1;

INSERT INTO members VALUES (6, 'John Doe', '2025-01-01', 'john@example.com');

ROLLBACK TO sp1;

**Lab 2:**

START TRANSACTION;

SAVEPOINT sp1;

INSERT INTO members VALUES (7, 'Jane Doe', '2025-02-01', 'jane@example.com');

COMMIT;

UPDATE members SET member\_name = 'Jane D.' WHERE member\_id = 7;

ROLLBACK;

### ****1. Introduction to SQL****

**Q:** 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.  
**A:**

CREATE DATABASE library\_db;

USE library\_db;

CREATE TABLE books (

book\_id INT PRIMARY KEY,

title VARCHAR(100),

author VARCHAR(100),

publisher VARCHAR(100),

year\_of\_publication INT,

price DECIMAL(10, 2)

);

INSERT INTO books VALUES

(1, 'Book A', 'Author X', 'Publisher A', 2010, 45.00),

(2, 'Book B', 'Author Y', 'Publisher B', 2015, 60.00),

(3, 'Book C', 'Author Z', 'Publisher C', 2020, 30.00),

(4, 'Book D', 'Author X', 'Publisher D', 2005, 25.00),

(5, 'Book E', 'Author Y', 'Publisher E', 2018, 50.00);

**Q:** Create a table members in library\_db with columns: member\_id, member\_name, date\_of\_membership, and email. Insert five records into this table.  
**A:**

CREATE TABLE members (

member\_id INT PRIMARY KEY,

member\_name VARCHAR(100),

date\_of\_membership DATE,

email VARCHAR(50)

);

INSERT INTO members VALUES

(1, 'Alice', '2019-03-12', 'alice@example.com'),

(2, 'Bob', '2021-07-01', 'bob@example.com'),

(3, 'Carol', '2018-05-20', 'carol@example.com'),

(4, 'Dave', '2023-01-11', 'dave@example.com'),

(5, 'Eve', '2020-11-25', 'eve@example.com');

### ****2. SQL Syntax****

**Q:** Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.  
**A:**

SELECT \* FROM members

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

ORDER BY date\_of\_membership;

**Q:** Write SQL queries to display the titles of books published by a specific author. Sort the results by year\_of\_publication in descending order.  
**A:**

SELECT title FROM books

WHERE author = 'Author X'

ORDER BY year\_of\_publication DESC;

### ****3. SQL Constraints****

**Q:** Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.  
**A:**

ALTER TABLE books

ADD CONSTRAINT chk\_price CHECK (price > 0);

**Q:** Modify the members table to add a UNIQUE constraint on the email column.  
**A:**

ALTER TABLE members

ADD CONSTRAINT unique\_email UNIQUE (email);

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

**Q:** Create a table authors with columns: author\_id, first\_name, last\_name, and country. Set author\_id as the primary key.  
**A:**

CREATE TABLE authors (

author\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

country VARCHAR(50)

);

**Q:** Create a table publishers with columns: publisher\_id, publisher\_name, contact\_number, and address. Set publisher\_id as primary key and contact\_number as unique.  
**A:**

CREATE TABLE publishers (

publisher\_id INT PRIMARY KEY,

publisher\_name VARCHAR(100),

contact\_number VARCHAR(20) UNIQUE,

address VARCHAR(255)

);

### ****5. ALTER Command****

**Q:** Add a new column genre to the books table. Update the genre for all existing records.  
**A:**

ALTER TABLE books ADD genre VARCHAR(50);

UPDATE books SET genre = 'Fiction' WHERE book\_id IN (1,2,3);

UPDATE books SET genre = 'Non-Fiction' WHERE book\_id IN (4,5);

**Q:** Modify the members table to increase the length of the email column to 100 characters.  
**A:**

ALTER TABLE members MODIFY email VARCHAR(100);

### ****6. DROP Command****

**Q:** Drop the publishers table from the database after verifying its structure.  
**A:**

DESC publishers;

DROP TABLE publishers;

**Q:** Create a backup of the members table and then drop the original members table.  
**A:**

CREATE TABLE members\_backup AS SELECT \* FROM members;

DROP TABLE members;

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

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

INSERT INTO authors (author\_id, first\_name, last\_name) VALUES (101, 'John', 'Smith');

INSERT INTO authors (author\_id, first\_name, last\_name) VALUES (102, 'Emily', 'Johnson');

INSERT INTO authors (author\_id, first\_name, last\_name) VALUES (103, 'Michael', 'Brown');

UPDATE authors SET last\_name = 'Williams' WHERE author\_id = 103;

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

DELETE FROM books WHERE price > 100;

### ****8. UPDATE Command****

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

UPDATE books SET year\_of\_publication = 2022 WHERE book\_id = 5;

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

UPDATE books SET price = price \* 1.10 WHERE year\_of\_publication < 2015;

### ****9. DELETE Command****

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

DELETE FROM members WHERE join\_date < '2020-01-01';

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

DELETE FROM books WHERE author IS NULL;

### ****10. Data Query Language (DQL)****

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

SELECT \* FROM books WHERE price BETWEEN 50 AND 100;

**Lab 5:**  
**Q:** Retrieve the list of books sorted by author in ascending order and limit the results to the top 3 entries.  
**A:**

SELECT \* FROM books ORDER BY author ASC LIMIT 3;

### ****11. Data Control Language (DCL)****

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

GRANT SELECT ON books TO librarian;

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

GRANT INSERT, UPDATE ON members TO admin;

### ****12. REVOKE Command****

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

REVOKE INSERT ON books FROM librarian;

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

REVOKE ALL PRIVILEGES ON members FROM admin;

### ****13. Transaction Control Language (TCL)****

**Lab 3:**  
**Q:** Use COMMIT after inserting multiple records into the books table, then make another insertion and perform a ROLLBACK.  
**A:**

BEGIN;

INSERT INTO books (book\_id, title, author, price) VALUES (201, 'SQL Basics', 'John Smith', 45);

INSERT INTO books (book\_id, title, author, price) VALUES (202, 'Advanced SQL', 'Emily Johnson', 75);

COMMIT;

INSERT INTO books (book\_id, title, author, price) VALUES (203, 'SQL Mastery', 'Michael Brown', 95);

ROLLBACK;

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

BEGIN;

SAVEPOINT before\_update;

UPDATE members SET status = 'inactive' WHERE last\_login < '2022-01-01';

UPDATE members SET membership\_type = 'basic' WHERE membership\_type = 'premium';

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;

### ****14. SQL Joins****

**Lab 3:**  
**Q:** Perform an INNER JOIN between books and authors tables to display the title of books and their respective authors' names.  
**A:**

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

FROM books

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

**Lab 4:**  
**Q:** 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.  
**A:**

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

FROM books

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

### ****15. SQL Group By****

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

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

FROM books

GROUP BY genre;

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

SELECT EXTRACT(YEAR FROM join\_date) AS join\_year, COUNT(\*) AS total\_members

FROM members

GROUP BY EXTRACT(YEAR FROM join\_date);

### ****16. SQL Stored Procedure****

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

CREATE PROCEDURE GetBooksByAuthor(IN authorName VARCHAR(100))

BEGIN

SELECT \* FROM books WHERE author = authorName;

END;

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

CREATE PROCEDURE GetBookPrice(IN b\_id INT)

BEGIN

SELECT price FROM books WHERE book\_id = b\_id;

END;

### ****17. SQL View****

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

CREATE VIEW book\_summary AS

SELECT title, author, price FROM books;

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

CREATE VIEW early\_members AS

SELECT \* FROM members WHERE join\_date < '2020-01-01';

### ****18. SQL Trigger****

**Lab 3:**  
**Q:** Create a trigger to automatically update the last\_modified timestamp of the books table whenever a record is updated.  
**A:**

CREATE TRIGGER update\_last\_modified

BEFORE UPDATE ON books

FOR EACH ROW

SET NEW.last\_modified = NOW();

**Lab 4:**  
**Q:** Create a trigger that inserts a log entry into a log\_changes table whenever a DELETE operation is performed on the books table.  
**A:**

CREATE TRIGGER log\_book\_deletion

AFTER DELETE ON books

FOR EACH ROW

INSERT INTO log\_changes (action\_type, book\_id, action\_time)

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

### ****19. Introduction to PL/SQL****

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

BEGIN

INSERT INTO books (book\_id, title, author, price)

VALUES (301, 'PLSQL Guide', 'Anna Scott', 59.99);

DBMS\_OUTPUT.PUT\_LINE('Book inserted successfully.');

END;

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

DECLARE

total\_books NUMBER;

BEGIN

SELECT COUNT(\*) INTO total\_books FROM books;

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

END;

### ****20. PL/SQL Syntax****

**Lab 3:**  
**Q:** Write a PL/SQL block to declare variables for book\_id and price, assign values, and display the results.  
**A:**

DECLARE

book\_id NUMBER := 101;

price NUMBER := 49.99;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || book\_id || ', Price: $' || price);

END;

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

DECLARE

CONSTANT discount\_rate NUMBER := 0.10;

original\_price NUMBER := 100;

final\_price NUMBER;

BEGIN

final\_price := original\_price - (original\_price \* discount\_rate);

DBMS\_OUTPUT.PUT\_LINE('Discounted price: $' || final\_price);

END;

### ****21. PL/SQL Control Structures****

**Lab 3:**  
**Q:** Write a PL/SQL block using IF-THEN-ELSE to check if a book's price is above $100 and print a message accordingly.  
**A:**

DECLARE

price NUMBER := 120;

BEGIN

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

DECLARE

CURSOR book\_cursor IS SELECT title, author, price FROM books;

v\_title books.title%TYPE;

v\_author books.author%TYPE;

v\_price books.price%TYPE;

BEGIN

FOR book\_record IN book\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Title: ' || book\_record.title ||

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

', Price: $' || book\_record.price);

END LOOP;

END;

### ****22. SQL Cursors****

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

DECLARE

CURSOR member\_cursor IS SELECT \* FROM members;

v\_member members%ROWTYPE;

BEGIN

OPEN member\_cursor;

LOOP

FETCH member\_cursor INTO v\_member;

EXIT WHEN member\_cursor%NOTFOUND;

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

', Name: ' || v\_member.name);

END LOOP;

CLOSE member\_cursor;

END;

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

DECLARE

CURSOR author\_books IS SELECT title FROM books WHERE author = 'John Smith';

v\_title books.title%TYPE;

BEGIN

OPEN author\_books;

LOOP

FETCH author\_books INTO v\_title;

EXIT WHEN author\_books%NOTFOUND;

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

END LOOP;

CLOSE author\_books;

END;

### ****23. Rollback and Commit Savepoint****

**Lab 3:**  
**Q:** Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.  
**A:**

START TRANSACTION;

INSERT INTO members (member\_id, name, join\_date) VALUES (401, 'David Green', '2025-07-01');

SAVEPOINT before\_update;

UPDATE members SET name = 'David G.' WHERE member\_id = 401;

ROLLBACK TO before\_update;

COMMIT;

**Lab 4:**  
**Q:** Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.  
**A:**

START TRANSACTION;

INSERT INTO books (book\_id, title, author, price) VALUES (501, 'Database Systems', 'Alan Turing', 60);

INSERT INTO books (book\_id, title, author, price) VALUES (502, 'AI and SQL', 'Ada Lovelace', 85);

COMMIT;

START TRANSACTION;

SAVEPOINT update\_point;

UPDATE books SET price = price + 10 WHERE book\_id = 501;

ROLLBACK TO update\_point;