Module-4
EXTRA LAB PRACTISE

1. Introduction to SQL

• **Lab 1**: 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.

```
-- Step 1: Create Database
CREATE DATABASE library_db;

-- Step 2: Use the Database
USE library_db;

-- Step 3: Create Table 'books'
CREATE TABLE books (
   book_id INT PRIMARY KEY,
   title VARCHAR(100),
   author VARCHAR(100),
   publisher VARCHAR(100),
   year_of_publication INT,
   price DECIMAL(8,2)
);

-- Step 4: Insert Records
INSERT INTO books (book_id, title, author, publisher, year_of_publication, price) VALUES
(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Scribner', 1925, 350.50),
(2, 'To Kill a Mockingbird', 'Harper Lee', 'J.B. Lippincott', 1960, 280.00),
(3, '1984', 'George Orwell', 'Secker & Warburg', 1949, 300.75),
(4, 'Pride and Prejudice', 'Jane Austen', 'T. Egerton', 1813, 250.00),
(5, 'The Alchemist', 'Paulo Coelho', 'HarperCollins', 1988, 400.00);
```

• **Lab 2**: Create a table `members` in `library_db` with columns: `member_id`, `member_name`, `date_of_membership`, and `email`. Insert five records into this table.

```
-- Step 1: Create Table 'members'
CREATE TABLE members (
   member_id INT PRIMARY KEY,
   member_name VARCHAR(100),
    date_of_membership DATE,
    email VARCHAR(100)
);

-- Step 2: Insert Records
INSERT INTO members (member_id, member_name, date_of_membership, email) VALUES
(101, 'Rahul Sharma', '2023-01-15', 'rahul.sharma@example.com'),
(102, 'Priya Mehta', '2023-02-10', 'priya.mehta@example.com'),
(103, 'Amit Patel', '2023-03-05', 'amit.patel@example.com'),
(104, 'Neha Singh', '2023-04-20', 'neha.singh@example.com'),
(105, 'Karan Joshi', '2023-05-18', 'karan.joshi@example.com');
```

2. SQL Syntax

• **Lab 1**: Retrieve all `members` who joined the library before 2022. Use appropriate SQL syntax with `WHERE` and `ORDER BY`.

```
-- Retrieve members with date_of_membership before 2022
SELECT *
FROM members
WHERE date_of_membership < '2022-01-01'
ORDER BY date_of_membership;
```

• **Lab 2**: Write SQL queries to display the titles of books published by a specific author. Sort the results by `year_of_publication` in descending order.

```
-- Replace 'George Orwell' with the desired author name
SELECT title
FROM books
WHERE author = 'George Orwell'
ORDER BY year_of_publication DESC;
```

3. SQL Constraints

• **Lab 1**: Add a `CHECK` constraint to ensure that the `price` of books in the `books` table is greater than 0.

```
-- Add CHECK constraint to ensure price > 0
ALTER TABLE books
ADD CONSTRAINT chk_price_positive
CHECK (price > 0);
```

• **Lab 2**: Modify the `members` table to add a `UNIQUE` constraint on the `email` column, ensuring that each member has a unique email address.

```
-- Add UNIQUE constraint to email column
ALTER TABLE members
ADD CONSTRAINT unique_email
UNIQUE (email);
```

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

• **Lab 1**: Create a table `authors` with the following columns: `author_id`, `first_name`, `last_name`, and `country`. Set `author_id` as the primary key.

```
CREATE TABLE authors (
   author_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   country VARCHAR(50)
);
```

• **Lab 2**: 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.

```
CREATE TABLE publishers (
   publisher_id INT PRIMARY KEY,
   publisher_name VARCHAR(100),
   contact_number VARCHAR(15) UNIQUE,
   address VARCHAR(200)
);
```

5. ALTER Command

• **Lab 1**: Add a new column `genre` to the `books` table. Update the `genre` for all existing records.

```
-- Step 1: Add new column
ALTER TABLE books ADD genre VARCHAR(50);
```

```
-- Step 2: Update all existing records with a default genre
UPDATE books
SET genre = 'Unknown';   -- You can replace 'Unknown' with any default value
```

• **Lab 2**: Modify the `members` table to increase the length of the `email` column to 100 characters.

```
-- Modify column size
ALTER TABLE members MODIFY email VARCHAR(100);
```

6. DROP Command

• **Lab 1**: Drop the `publishers` table from the database after verifying its structure.

```
-- Check structure of table
DESC publishers;

-- Drop the table
DROP TABLE publishers;
```

• **Lab 2**: Create a backup of the `members` table and then drop the original `members` table.

```
-- Create backup
CREATE TABLE members_backup AS SELECT * FROM members;

-- Drop original table
DROP TABLE members;
```

7. Data Manipulation Language (DML)

• **Lab 1**: Insert three new authors into the `authors` table, then update the last name of one of the authors.

```
-- Insert new authors
INSERT INTO authors (author_id, first_name, last_name, country)
VALUES (101, 'John', 'Smith', 'USA');

INSERT INTO authors (author_id, first_name, last_name, country)
VALUES (102, 'Priya', 'Sharma', 'India');

INSERT INTO authors (author_id, first_name, last_name, country)
```

```
VALUES (103, 'David', 'Brown', 'UK');

-- Update last name of one author
UPDATE authors
SET last_name = 'Johnson'
WHERE author_id = 101;
```

• **Lab 2**: Delete a book from the `books` table where the `price` is higher than $100.

```
DELETE FROM books
WHERE price > 100;
```

8. UPDATE Command

• **Lab 1**: Update the `year_of_publication` of a book with a specific `book_id`.

```
UPDATE books
SET year_of_publication = 2022
WHERE book_id = 5;
```

• **Lab 2**: Increase the `price` of all books published before 2015 by 10%.

```
UPDATE books
SET price = price * 1.10
WHERE year_of_publication < 2015;
```

9. DELETE Command

• **Lab 1**: Remove all members who joined before 2020 from the `members` table.

```
DELETE FROM members
WHERE join_year < 2020;
```

• **Lab 2**: Delete all books that have a `NULL` value in the `author` column.

```
DELETE FROM books
WHERE author IS NULL;
```

10. Data Query Language (DQL)

• **Lab 1**: Write a query to retrieve all `books` with `price` between $50 and $100.

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

• **Lab 2**: Retrieve the list of `books` sorted by `author` in ascending order and limit the results to the top 3 entries.

```
SELECT * FROM books
ORDER BY author ASC
FETCH FIRST 3 ROWS ONLY;   -- (Oracle / SQL standard)

-- For MySQL use:
-- SELECT * FROM books ORDER BY author ASC LIMIT 3;
```

11. Data Control Language (DCL)

• **Lab 1**: Grant `SELECT` permission to a user named `librarian` on the `books` table.

```
GRANT SELECT ON books TO librarian;
```

• **Lab 2**: Grant `INSERT` and `UPDATE` permissions to the user `admin` on the `members` table.

```
GRANT INSERT, UPDATE ON members TO admin;
```

12. REVOKE Command

• **Lab 1**: Revoke the `INSERT` privilege from the user `librarian` on the `books` table.

```
REVOKE INSERT ON books FROM librarian;
```

• **Lab 2**: Revoke all permissions from user `admin` on the `members` table.

```
REVOKE ALL PRIVILEGES ON members FROM admin;
```

13. Transaction Control Language (TCL)

• **Lab 1**: Use `COMMIT` after inserting multiple records into the `books` table, then make another insertion and perform a `ROLLBACK`.

```
INSERT INTO books VALUES (201, 'Book A', 'Author X', 50, 2018);
INSERT INTO books VALUES (202, 'Book B', 'Author Y', 70, 2019);
COMMIT;

INSERT INTO books VALUES (203, 'Book C', 'Author Z', 90, 2020);
ROLLBACK;
```

• **Lab 2**: Set a `SAVEPOINT` before making updates to the `members` table, perform some updates, and then roll back to the `SAVEPOINT`.

```
SAVEPOINT before_update;

UPDATE members SET email = 'newmail@test.com' WHERE member_id = 5;
UPDATE members SET email = 'another@test.com' WHERE member_id = 6;

ROLLBACK TO before_update;
```

14. SQL Joins

• **Lab 1**: Perform an `INNER JOIN` between `books` and `authors` tables to display the `title` of books and their respective authors' names.

```
SELECT b.title, a.first_name, a.last_name
FROM books b
INNER JOIN authors a ON b.author_id = a.author_id;
```

• **Lab 2**: 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.

```
SELECT b.title, a.first_name, a.last_name
FROM books b
FULL OUTER JOIN authors a ON b.author_id = a.author_id;
```

15. SQL Group By

• **Lab 1**: Group `books` by `genre` and display the total number of books in each genre.

```
SELECT genre, COUNT(*) AS total_books
FROM books
GROUP BY genre;
```

• **Lab 2**: Group `members` by the year they joined and find the number of members who joined each year.

```
SELECT join_year, COUNT(*) AS total_members
FROM members
GROUP BY join_year;
```

16. SQL Stored Procedure

• **Lab 1**: Write a stored procedure to retrieve all `books` by a particular `author`.

```
CREATE PROCEDURE getBooksByAuthor(p_author_id INT) AS
BEGIN
  SELECT title FROM books WHERE author_id = p_author_id;
END;
```

• **Lab 2**: Write a stored procedure that takes `book_id` as an argument and returns the `price` of the book.

```
CREATE PROCEDURE getBookPrice(p_book_id INT) AS
  v_price NUMBER;
BEGIN
  SELECT price INTO v_price FROM books WHERE book_id = p_book_id;
  DBMS_OUTPUT.PUT_LINE('Price: ' || v_price);
END;
```

17. SQL View

• **Lab 1**: Create a view to show only the `title`, `author`, and `price` of books from the `books` table.

```
CREATE VIEW book_view AS
```

```
SELECT title, author_id, price FROM books;
```

- **Lab 2**: Create a view to display `members` who joined before 2020.

```
CREATE VIEW old_members AS
SELECT * FROM members WHERE join_year < 2020;
```

18. SQL Trigger

- **Lab 1**: Create a trigger to automatically update the `last_modified` timestamp of the `books` table whenever a record is updated.

```
CREATE OR REPLACE TRIGGER update_timestamp
BEFORE UPDATE ON books
FOR EACH ROW
BEGIN
  :NEW.last_modified := SYSDATE;
END;
```

- **Lab 2**: Create a trigger that inserts a log entry into a `log_changes` table whenever a `DELETE` operation is performed on the `books` table.

```
CREATE OR REPLACE TRIGGER log_delete
AFTER DELETE ON books
FOR EACH ROW
BEGIN
  INSERT INTO log_changes (table_name, action, action_date)
  VALUES ('books', 'DELETE', SYSDATE);
END;
```

19. Introduction to PL/SQL

- **Lab 1**: Write a PL/SQL block to insert a new `book` into the `books` table and display a confirmation message.

```
BEGIN
  INSERT INTO books VALUES (301, 'New Book', 'Author M', 120, 2023);
  DBMS_OUTPUT.PUT_LINE('Book inserted successfully!');
END;
```

• **Lab 2**: Write a PL/SQL block to display the total number of books in the `books` table.

```
DECLARE
  total_books NUMBER;
BEGIN
  SELECT COUNT(*) INTO total_books FROM books;
  DBMS_OUTPUT.PUT_LINE('Total books: ' || total_books);
END;
```

20. PL/SQL Syntax

• **Lab 1**: Write a PL/SQL block to declare variables for `book_id` and `price`, assign values, and display the results.

```
DECLARE
  v_book_id NUMBER := 101;
  v_price NUMBER := 200;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Book ID: ' || v_book_id || ' Price: ' || v_price);
END;
```

• **Lab 2**: Write a PL/SQL block using `constants` and perform arithmetic operations on book prices.

```
DECLARE
  c_tax CONSTANT NUMBER := 10;
  v_price NUMBER := 200;
  v_final NUMBER;
BEGIN
  v_final := v_price + c_tax;
  DBMS_OUTPUT.PUT_LINE('Final Price: ' || v_final);
END;
```

21. PL/SQL Control Structures

• **Lab 1**: Write a PL/SQL block using `IF-THEN-ELSE` to check if a book's price is above $100 and print a message accordingly.

```
DECLARE
  v_price NUMBER := 150;
BEGIN
```

```
  IF v_price > 100 THEN
    DBMS_OUTPUT.PUT_LINE('Price is above $100');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Price is $100 or below');
  END IF;
END;
```

• **Lab 2**: Use a `FOR LOOP` in PL/SQL to display the details of all books one by one.

```
DECLARE
  CURSOR book_cur IS SELECT * FROM books;
  v_book books%ROWTYPE;
BEGIN
  FOR v_book IN book_cur LOOP
    DBMS_OUTPUT.PUT_LINE(v_book.title || ' - ' || v_book.price);
  END LOOP;
END;
```

22. SQL Cursors

• **Lab 1**: Write a PL/SQL block using an explicit cursor to fetch and display all records from the `members` table.

```
DECLARE
  CURSOR mem_cur IS SELECT * FROM members;
  v_member members%ROWTYPE;
BEGIN
  OPEN mem_cur;
  LOOP
    FETCH mem_cur INTO v_member;
    EXIT WHEN mem_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_member.member_id || ' - ' || v_member.email);
  END LOOP;
  CLOSE mem_cur;
END;
```

• **Lab 2**: Create a cursor to retrieve books by a particular author and display their titles.

```
DECLARE
  CURSOR book_cur IS SELECT title FROM books WHERE author_id = 102;
  v_title books.title%TYPE;
BEGIN
```

```
  OPEN book_cur;
  LOOP
    FETCH book_cur INTO v_title;
    EXIT WHEN book_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_title);
  END LOOP;
  CLOSE book_cur;
END;
```

23. Rollback and Commit Savepoint

• **Lab 1**: Perform a transaction that includes inserting a new `member`, setting a `SAVEPOINT`, and rolling back to the savepoint after making updates.

```
INSERT INTO members VALUES (501, 'Sam', 'sam@mail.com', 2022);
SAVEPOINT sp1;

UPDATE members SET email = 'wrong@mail.com' WHERE member_id = 501;
ROLLBACK TO sp1;
```

• **Lab 2**: Use `COMMIT` after successfully inserting multiple books into the `books` table, then use `ROLLBACK` to undo a set of changes made after a savepoint.

```
INSERT INTO books VALUES (401, 'Book X', 'Author A', 80, 2017);

INSERT INTO books VALUES (402, 'Book Y', 'Author B', 95, 2018);

COMMIT;

SAVEPOINT sp2;

UPDATE books SET price = 200 WHERE book_id = 401;

ROLLBACK TO sp2;
```