Module\_5 LAB EXERCISES

**Introduction to SQL:**

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: To create a new database named school\_db and a table called students with the specified columns, you would use SQL commands.

First, create the database:

CREATE DATABASE school\_db;

Next, select the newly created database to work within it:

USE school\_db;

Finally, create the students table with the defined columns and their respective data types. It is recommended to set student\_id as the primary key for unique identification.

CREATE TABLE students (  
 student\_id INT PRIMARY KEY,  
 student\_name VARCHAR(100),  
 age INT,  
 class VARCHAR(50),  
 address VARCHAR(255)  
);

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

Ans: To insert five records into a students table and then retrieve all records from it, the following SQL statements are used:

1. Inserting Five Records:

Assuming a students table with columns like student\_id (integer), name (varchar), and grade (integer), the INSERT INTO statement is used:

INSERT INTO students (student\_id, name, grade) VALUES  
(1, 'Alice', 95),  
(2, 'Bob', 88),  
(3, 'Charlie', 72),  
(4, 'Diana', 91),  
(5, 'Eve', 85);

2. Retrieving All Records:

To retrieve all records from the students table, the SELECT statement is used:

SELECT \* FROM students;

**SQL Syntax:**

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

Ans: To retrieve specific columns, such as student\_name and age, from a table named students, the SELECT statement in SQL is used.

SELECT student\_name, age  
FROM students;

This query will return a result set containing only the student\_name and age columns for all records present in the students table. If you need to retrieve all columns, you would use the asterisk (\*) wildcard instead of specifying individual column names:

SELECT \*  
FROM students;

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

Ans: To retrieve all students whose age is greater than 10 from a table named students, the following SQL query can be used:

SELECT \*  
FROM students  
WHERE age > 10;

**SQL Constraints:**

1: Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

Ans: To create a table named teachers with the specified columns and constraints, the following SQL CREATE TABLE statement can be used:

CREATE TABLE teachers (  
 teacher\_id INT PRIMARY KEY,  
 teacher\_name VARCHAR(255) NOT NULL,  
 subject VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE  
);

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

Ans: To implement a FOREIGN KEY constraint relating teacher\_id in the students table to the teachers table, assuming teacher\_id is the primary key in the teachers table, you can use the ALTER TABLE statement.

1. Adding the teacher\_id column to the students table (if it doesn't exist):

If your students table does not already have a teacher\_id column, you need to add it first, ensuring its data type matches the teacher\_id in the teachers table.

ALTER TABLE students  
ADD COLUMN teacher\_id INT; -- Adjust data type as needed (e.g., BIGINT, UUID)

2. Adding the FOREIGN KEY constraint:

Once the teacher\_id column exists in the students table, you can add the FOREIGN KEY constraint.

ALTER TABLE students  
ADD CONSTRAINT fk\_teacher\_student *-- Optional: Give a meaningful name to the constraint*  
FOREIGN KEY (teacher\_id)  
REFERENCES teachers(teacher\_id);

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

1: Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.

Ans: To create a table named courses with the specified columns and primary key, the following SQL CREATE TABLE statement can be used:

CREATE TABLE courses (  
 course\_id INT PRIMARY KEY,  
 course\_name VARCHAR(255),  
 course\_credits INT  
);

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

Ans: To create a database named "university\_db" using the CREATE command in SQL, the following syntax is used:

CREATE DATABASE university\_db;

This statement instructs the database management system (DBMS) to establish a new database container named university\_db.

ALTER Command:

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

Ans: To add a column named course\_duration to the courses table using the ALTER TABLE command, the following SQL statement can be used:

ALTER TABLE courses  
ADD COLUMN course\_duration INT;

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

Ans: To drop the credits column from the course table in a database, the ALTER TABLE statement with the DROP COLUMN clause is used.

ALTER TABLE course  
DROP COLUMN credits;

DROP Command:

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

Ans: To drop the teachers table from the school\_db database, execute the SQL command DROP TABLE teachers; within a SQL client connected to the school\_db database.

DROP TABLE teachers;

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

Ans: To drop the students table from the school\_db database and verify its removal, execute the following SQL commands:

USE school\_db;  
DROP TABLE students;

To verify the table's removal, query the database's information schema or list the tables within school\_db:

USE school\_db;  
SHOW TABLES;

Alternatively, query the information\_schema.tables view:

SELECT table\_name  
FROM information\_schema.tables  
WHERE table\_schema = 'school\_db' AND table\_name = 'students';

Data Manipulation Language (DML):

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

Ans: To insert three records into a courses table using the INSERT command, the following SQL statement can be used, assuming the courses table has columns for course\_id, course\_name, and credits:

INSERT INTO courses (course\_id, course\_name, credits) VALUES  
(101, 'Introduction to Programming', 3),  
(102, 'Database Management Systems', 4),  
(103, 'Web Development Fundamentals', 3);

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

Ans: To update the course duration for a specific course, use the SQL UPDATE command with a SET clause to specify the new duration and a WHERE clause to identify the particular course by its ID or another unique identifier. For example: UPDATE Courses SET duration = 'new\_duration' WHERE course\_id = 'specific\_course\_id'; .

Example

Imagine you have a table named Courses with a course\_id and a duration column, and you want to change the duration of the course with course\_id 'CS101' to '4 months'.

UPDATE Courses

SET duration = '4 months'

WHER course\_id 'CS101' ;

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

Ans: To delete a course with a specific course\_id from the courses table using the DELETE command, the following SQL statement can be used:

DELETE FROM courses  
WHERE course\_id = 'your\_course\_id';

Data Query Language (DQL):

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

Ans:

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

Ans:

3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Ans:

Data Control Language (DCL):

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

Ans:

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

Ans:

Transaction Control Language (TCL):

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

Ans:

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

Ans:

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

Ans:

SQL Joins:

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

Ans:

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

Ans:

SQL Group By:

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

Ans:

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

Ans:

SQL Stored Procedure:

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

Ans:

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

Ans:

SQL View:

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

Ans:

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

Ans:

SQL Triggers:

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

Ans:

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

Ans:

Introduction to PL/SQL:

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

Ans:

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

Ans:

PL/SQL Control Structures:

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

Ans:

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

Ans:

SQL Cursors:

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

Ans:

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

Ans:

Rollback and Commit Savepoint:

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

Ans:

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

Ans: