***1. 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.**

-- Step 1: Create the database

CREATE DATABASE school\_db;

-- Step 2: Use the new database

USE school\_db;

-- Step 3: Create the students table

CREATE TABLE students (

student\_id INT AUTO\_INCREMENT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

age INT,

class VARCHAR(50),

address VARCHAR(255)

);

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

-- Insert five records with Indian names into the students table

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

VALUES

('Aarav Sharma', 15, '10th Grade', '12 Nehru Nagar, Delhi'),

('Anaya Patel', 14, '9th Grade', '45 MG Road, Ahmedabad'),

('Vivaan Mehta', 16, '11th Grade', '78 Park Street, Kolkata'),

('Diya Reddy', 15, '10th Grade', '19 Jubilee Hills, Hyderabad'),

('Krishna Nair', 17, '12th Grade', '101 Marine Drive, Mumbai');

-- Retrieve all records from the students table

SELECT \* FROM students;

***2. SQL Syntax***

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

SELECT student\_name, age FROM students;

|  |  |
| --- | --- |
| **student\_name** | **age** |
| Aarav Sharma | 15 |
| Anaya Patel | 14 |
| Vivaan Mehta | 16 |
| Diya Reddy | 15 |
| Krishna Nair | 17 |

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

SELECT \* FROM students WHERE age > 10;

***3. 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).**

CREATE TABLE teachers (

teacher\_id INT AUTO\_INCREMENT PRIMARY KEY,

teacher\_name VARCHAR(100) NOT NULL,

subject VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE

);

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

ALTER TABLE students ADD teacher\_id INT;

ALTER TABLE students ADD CONSTRAINT fk\_teacher FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id);

***4. 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.**

CREATE TABLE courses (

course\_id INT AUTO\_INCREMENT PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

course\_credits INT

);

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

CREATE DATABASE university\_db;

USE university\_db;

***5. ALTER Command***

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

ALTER TABLE courses ADD course\_duration VARCHAR(50);

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

ALTER TABLE courses DROP COLUMN course\_credits;

***6. DROP Command***

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

DROP TABLE school\_db.teachers;

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

DROP TABLE school\_db.students;

You can list the tables in the school\_db database with the following command:

SHOW TABLES FROM school\_db;

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

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

INSERT INTO courses (course\_name, course\_duration) VALUES ('Introduction to Programming', '6 months'),('Database Management Systems', '1 year'),('Web Development', '6 months');

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

UPDATE courses SET course\_duration = '1 year' WHERE course\_name = 'Introduction to Programming';

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

DELETE FROM courses WHERE course\_id = 1;

***8. Data Query Language (DQL)***

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

SELECT \* FROM courses;

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

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

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

SELECT \* FROM courses LIMIT 2;

**9. Data Control Language (DCL)**

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

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

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

Grant user1 permission to SELECT from the courses table

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

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

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

Grant the INSERT permission to user2

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

***10. Transaction Control Language (TCL)***

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

INSERT INTO courses (course\_name, course\_duration) VALUES ('Data Structures', '6 months'),('Machine Learning', '1 year'),('Digital Marketing', '3 months');

COMMIT;

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

INSERT INTO courses (course\_name, course\_duration)VALUES ('Cloud Computing', '6 months'),

('Cyber Security', '1 year');

ROLLBACK;

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

SAVEPOINT before\_update;

UPDATE courses SET course\_duration = '2 years' WHERE course\_name = 'Machine Learning';

ROLLBACK TO SAVEPOINT before\_update;

***11. SQL Joins***

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

CREATE TABLE departments (

department\_id INT AUTO\_INCREMENT PRIMARY KEY,

department\_name VARCHAR(100) NOT NULL

);

CREATE TABLE employees (

employee\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_name VARCHAR(100) NOT NULL,

department\_id INT,

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

);

-- Insert departments

INSERT INTO departments (department\_name) VALUES ('HR'), ('IT'), ('Finance');

-- Insert employees

INSERT INTO employees (employee\_name, department\_id) VALUES ('Alice', 1), ('Bob', 2), ('Charlie', 3), ('David', 2);

Perform an INNER JOIN

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

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

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

***12. SQL Group By***

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

SELECT departments.department\_name, COUNT(employees.employee\_id) AS employee\_count FROM employees INNER JOIN departments ON employees.department\_id = departments.department\_id GROUP BY departments.department\_name;

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

SELECT departments.department\_name, AVG(employees.salary) AS average\_salary FROM employees INNER JOIN departments ON employees.department\_id = departments.department\_id GROUP BY departments.department\_name;

***13. SQL Stored Procedure***

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

DELIMITER $$

CREATE PROCEDURE GetEmployeesByDepartment (IN department\_name VARCHAR(100))

BEGIN

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

END $$

DELIMITER ;

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

DELIMITER $$

CREATE PROCEDURE GetCourseDetails (IN course\_id INT)

BEGIN

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

WHERE course\_id = course\_id;

END $$

DELIMITER ;

***14. SQL View***

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

CREATE VIEW EmployeeDepartmentView AS

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

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

-- Drop the existing view if it exists

DROP VIEW IF EXISTS EmployeeDepartmentView;

-- Create the modified view with salary condition

CREATE VIEW EmployeeDepartmentView AS

SELECT employees.employee\_id, employees.employee\_name, employees.salary, departments.department\_name FROM employees INNER JOIN departments ON employees.department\_id = departments.department\_id WHERE employees.salary >= 50000;

***15. SQL Triggers***

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

CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

action VARCHAR(50),

employee\_id INT,

employee\_name VARCHAR(100),

salary DECIMAL(10, 2),

log\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER log\_employee\_insert

AFTER INSERT ON employees FOR EACH ROW

BEGIN

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

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

END $$

DELIMITER ;

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

VALUES (4, 'David', 60000, 2);

SELECT \* FROM employee\_log;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **log\_id** | **action** | **employee\_id** | **employee\_name** | **salary** | **log\_timestamp** |
| 1 | INSERT | 4 | David | 60000 | 2025-04-21 12:34:56 |

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

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

DELIMITER $$

CREATE TRIGGER update\_employee\_last\_modified

BEFORE UPDATE ON employees FOR EACH ROW

BEGIN

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END $$

DELIMITER ;

UPDATE employees SET employee\_name = 'John Doe', salary = 65000

WHERE employee\_id = 1;

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

WHERE employee\_id = 1;

|  |  |  |  |
| --- | --- | --- | --- |
| **employee\_id** | **employee\_name** | **salary** | **last\_modified** |
| 1 | John Doe | 65000 | 2025-04-21 12:45:00 |

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

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

DECLARE

total\_employees NUMBER; -- Variable to store the total number of employees

BEGIN

-- Query to get the total number of employees

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

-- Print the total number of employees

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

END;

/

SET SERVEROUTPUT ON;

Total number of employees: 150

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

DECLARE

total\_sales NUMBER; -- Variable to store the total sales amount

BEGIN

-- Query to calculate the total sales by summing the order\_total

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

-- Print the total sales amount

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

END;

/

SET SERVEROUTPUT ON;

Total sales: 500000

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

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

DECLARE

v\_employee\_id employees.employee\_id%TYPE := 101; -- Example employee ID

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

BEGIN

-- Retrieve the department ID for the given employee

SELECT department\_id INTO v\_department\_id

FROM employees

WHERE employee\_id = v\_employee\_id;

-- Check if the employee belongs to department 10

IF v\_department\_id = 10 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee is in Department 10.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee is not in Department 10.');

END IF;

END;

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

BEGIN

-- Iterate through employee records using FOR LOOP

FOR emp\_record IN (SELECT employee\_name FROM employees) LOOP

-- Display the employee name

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_record.employee\_name);

END LOOP;

END;

***18. SQL Cursors***

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

DECLARE

-- Declare a cursor to retrieve employee details

CURSOR emp\_cursor IS

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

-- Variables to hold employee details

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

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

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

BEGIN

-- Open the cursor

OPEN emp\_cursor;

-- Fetch and display employee details one by one

LOOP

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

EXIT WHEN emp\_cursor%NOTFOUND; -- Exit loop when no more rows

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_employee\_id ||

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

', Department ID: ' || v\_department\_id);

END LOOP;

-- Close the cursor

CLOSE emp\_cursor;

END;

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

DECLARE

-- Declare a cursor to select all courses

CURSOR course\_cursor IS

SELECT course\_name FROM courses;

-- Variable to hold each course name

course\_name courses.course\_name%TYPE;

BEGIN

-- Open the cursor

OPEN course\_cursor;

-- Fetch and display each course one by one

LOOP

FETCH course\_cursor INTO course\_name;

EXIT WHEN course\_cursor%NOTFOUND; -- Exit loop if no more rows

DBMS\_OUTPUT.PUT\_LINE('Course: ' || course\_name);

END LOOP;

-- Close the cursor

CLOSE course\_cursor;

END;

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

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

BEGIN

-- Step 1: Insert a record (first operation)

INSERT INTO employees (employee\_id, name, department\_id) VALUES (101, 'John Doe', 10);

-- Step 2: Create a SAVEPOINT

SAVEPOINT before\_rollback;

-- Step 3: Insert another record (second operation)

INSERT INTO employees (employee\_id, name, department\_id) VALUES (102, 'Jane Smith', 20);

-- Step 4: Rollback to the SAVEPOINT (undo second operation)

ROLLBACK TO before\_rollback;

-- Step 5: Commit the first operation (first insert remains)

COMMIT;

END;

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

BEGIN

-- Step 1: First operation

UPDATE employees SET salary = salary + 1000 WHERE department\_id = 10;

-- Step 2: Set a SAVEPOINT

SAVEPOINT before\_rollback;

-- Step 3: Second operation (which might fail)

UPDATE employees SET salary = salary + 500 WHERE department\_id = 20;

-- Step 4: Commit part of the transaction

COMMIT;

-- Step 5: Rollback to savepoint (undo second operation)

ROLLBACK TO before\_rollback;

-- Final Step: Commit the remaining changes (if any)

COMMIT;

END;