Module 4 – Introduction to DBMS

1. Introduction to DBMS

- **Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.**

  - create database school_db

  - create table student(s_id int , s_name varchar(20) , age int , class int , address varchar(20) , t_id int , foreign key (t_id) references teacher(t_id));

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

  - insert into teacher values (1 , 'Dhruv' , 'C++' , 'abc@gmail.com'),
    (2 , 'Dharmistha' , 'Python' , 'xyz@gmail.com'),
    (3 , 'Muskan' , 'C' , 'axc@gmail.com'),
    (4 , 'Divya' , 'C++' , 'xbz@gmail.com'),
    (5 , 'Anjali' , 'Python' , 'aab@gmail.com');

2. SQL Syntax

- **Write SQL queries to retrieve specific columns (student_name and age) from the students table.**

  - select s_name , age from student

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

  - select * from student where age > 10

Module 4 – Introduction to DBMS

3. SQL Constraints

- **Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).**

  - create table teacher(t_id int primary key , t_name varchar(20) NOT NULL , sub varchar(20) not null , email varchar(20) unique**)**

- **Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.**

  - create table student(s_id int , s_name varchar(20) , age int , class int , address varchar(20) , t_id int , foreign key (t_id) references teacher(t_id));

4. Main SQL Commands and Sub – Commands (DDL)

- **Create a table courses with columns: course_id , course_name , and course_credits. Set the course_id as the primary key.**

  - create table courses(c_id int primary key , c_name varchar(20) , course_credits varchar(20))

- **Use the CREATE command to create a database university_db.**

  - create database university_db

Module 4 – Introduction to DBMS

5. ALTER Command

- **Modify the courses table by adding a column course_duration using the ALTER command.**
    - alter table courses add course_duration varchar(20)
- **Drop the course_credits column from the courses table.**
    - alter table courses drop column course_duration

6. DROP Command

- **Drop the teachers table from the school_db database.**
    - drop table teacher
- **Drop the students table from the school_db database and verify that the table has been removed.**
    - drop table student

Module 4 – Introduction to DBMS

7. Data Manipulation Language (DML)

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

  - insert into courses values (11 , 'Frontend Development' , '6 months'),
    (12 , 'Data Analysis' , '1 yr '),
    (13 , 'Backend Development' , '6 months');

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

  - update courses set course_duration = '6 yr' where c_id = '11'

8. Data Query Language (DQL)

- **Retrieve all courses from the courses table using the SELECT statement**.

  - select * from courses

- **Sort the courses based on course_duration in descending order using ORDER BY.**

  - select * from courses order by course_duration desc

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

  - select * from courses limit 2

Module 4 – Introduction to DBMS

9. Data Control Language (DCL)

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

  - create user user1@localhost identified by 'Shivam@0910'
  - create user user2 identified by 'Shivam@0910'
  - grant select on school_db.courses to user1@localhost;

- **Revoke the INSERT permission from user1 and give it to user2**

  - revoke insert on school_db.courses from user1@localhost;
  - grant insert on school_db.courses to user2;

10. Transaction Control Language (TCL)

- **Insert a few rows into the courses table and use COMMIT to save the changes**

  - insert into courses values (14 , 'A.I' , '1 yr'),
    (15 , 'Machine Learning' , '1.5 yr');
    Commit;

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

  - insert into courses values (16 , 'Web Development' , '1 yr'),
    (17 , 'FullStack' , '1.5 yr');
    rollback;

Module 4 – Introduction to DBMS

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

  - start transaction;
    savepoint s1;
    insert into courses values(18 , 'Graphics Design' , '1 yr')
    rollback to savepoint s1;
    select * from courses

11. Data Control Language (DCL)

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

  - select e_id , emp_name , dept_name from employee , department where employee.d_id = department.d_id order by e_id

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

  - select * from employee left join department on employee.d_id = department.d_id

12. SQL Group By

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

  - select dept_name , count(e_id) from employee , department where employee.d_id = department.d_id group by dept_name

Module 4 – Introduction to DBMS

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

  - select dept_name , avg(salary) from employee , department where employee.d_id = department.d_id group by dept_name

13. SQL Stored Procedure

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

  - DELIMITER //
    CREATE PROCEDURE get_employees_by_department(IN p_dept_name VARCHAR(50))
    BEGIN
      SELECT e.e_id, e.emp_name, e.salary, d.dept_name
      FROM employee e
    JOIN department d ON e.d_id = d.d_id
    WHERE d.dept_name = p_dept_name;
    END //

  - call get_employees_by_department('Backend');

- **Write a stored procedure that accepts course_id as input and returns the course details.**

  - Delimiter //
    create procedure course_info(in courseid int)
    begin
        select * from courses where c_id = course;
    end //

  - call get_course_info(12);

Module 4 – Introduction to DBMS

14. SQL View

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

  - create view emp_view
    as
    select e_id , emp_name , salary , dept_name
    from employee , department
    where employee.d_id = department.d_id;

  - select * from emp_view;

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

  - create view e_view
    as
    select e_id , emp_name , salary , dept_name
    from employee , department
    where employee.d_id = department.d_id and salary > 50000;

  - select * from e_view;

15. SQL Triggers

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

  - delimiter //
    create trigger log after insert on employee
    for each row
    begin
    insert into employee(emp_name , salary , d_id)
    values(new.emp_name , new.salary , new.d_id);
    end //

  - insert into Employee values(016,'Sonit',60000,102);

Module 4 – Introduction to DBMS

- **Create a trigger to update the last_modified timestamp whenever an employee record is updated.**

  - alter table employee add column last_modified timestamp
  - default current_timestamp ;

  - delimiter //
    create trigger last_edited before update on employee
    for each row
    begin
    set new.last_modified = current_timestamp;
    end //

  - update employee set emp_name='Rahul' where emp_id=104;

16. Introduction PL/SQL

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

  - delimiter //
    create procedure total_employee()
    begin
    declare total int;
    select count(e_id) into total from employee;
    select concat('total employee: ',total) as Result;
    end //

  - call total_employee();

Module 4 – Introduction to DBMS

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

  - ```
    delimiter //
    create procedure cal_total_sale()
    begin
    declare total int;
    select sum(sales) into total from orders;
    select concat('total sales:- ',total) as result;
    end //
    ```

  - ```
    call cal_total_sale();
    ```

17. PL/SQL Control Structures

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

  - ```
    delimiter //
    create procedure dept_check( employee_id int)
    begin
    declare emp_dept int;
    select dept_id into emp_dept from employee where e_id =
    employee_id;
    if emp_dept = 1 then
    select concat('Employee',employee_id,'works in Backend
    department') as message;
    end //
    ```

  - ```
    call dept_check(101);
    ```

Module 4 – Introduction to DBMS

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

  - delimiter //
    create procedure loop_ex()
    begin
    declare done int default false;
    declare empname varchar(20);
    declare emp_cur cursor for
    select emp_name from employee;
    declare continue handler for not found set done = true;
    open emp_cur;
    emp_loop:loop
    fetch emp_cur into empname;
    if done then
    leave emp_loop;
    end if;
    select empname as 'employee name ';
    end loop;
    close emp_cur;
    end //
    delimiter ;

  - call loop_ex();

Module 4 – Introduction to DBMS

18. SQL Cursors

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

```
delimiter //
create procedure cur2()
begin
declare c_emp_id int;
declare c_emp_name varchar(20);
declare c_emp_sal int;
declare done int default false;
declare emp_cur cursor for
select e_id , emp_name , salary from employee;
declare continue handler for not found set done=true;
open emp_cur;
read_loop:loop
fetch emp_cur into c_emp_id,c_emp_name,c_emp_sal;
if done then
leave read_loop;
end if;
select concat('ID= ',c_emp_id,' Name= ',c_emp_name,'
Salary=',c_emp_sal) as emp_details;
end loop;
end //
delimiter ;
```

```
call cur2();
```

Module 4 – Introduction to DBMS

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

```
delimiter //
create procedure cur_3()
begin
declare done int default false;
declare c_course_id int;
declare c_course_name varchar(30);
declare c_course_duration varchar(20);
declare course_cur cursor for
select c_id , c_name , course_duration from courses;
declare continue handler for not found set done=true;
create temporary table if not exists tmp_course(cid int ,
c_name1 varchar(20) , c_dura varchar(20));
truncate table tmp_course;
open course_cur;
read_loop:loop
fetch course_cur into
c_course_id , c_course_name , c_course_duration;
if done then
leave read_loop;
end if;
insert into tmp_course
values(c_course_id , c_course_name , c_course_duration);
end loop;
close course_cur;
select * from tmp_course;
end //
delimiter ;
```

```
call cur_3();
```

Module 4 – Introduction to DBMS

19. RollBack and Commit SavePoint

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

```
start transaction;

select * from employee;
insert into employee(emp_name , d_id , salary) values ('om',4,150000);
insert into employee(emp_name , d_id , salary) values ('Anuj',2,50000);

savepoint sp1;

insert into employee(emp_name , d_id , salary) values ('rudra',1,20000);
insert into employee(emp_name , d_id , salary) values ('prince',3,10000);

rollback to sp1;
commit;
```

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

```
start transaction;

insert into employee(emp_name , d_id , salary) values ('rudra',1,20000);
insert into employee(emp_name , d_id , salary) values ('prince',3,10000);
savepoint sp2;
commit;

insert into employee(emp_name , d_id , salary) values ('rahul',2,20000);
insert into employee(emp_name , d_id , salary) values ('pandya',4,10000);
rollback
select * from employee;
```