Lab Manual

of

Database Management Systems

Subject Code: CS 313
Class: ICD Vth Semester

Prepared by:

Dr. Jagdeep Singh

Assistant Professor (CSE)

Department
of
Computer Science and Engineering

Sant Longowal Institute of Engineering and Technology, Longowal, India

October 2024

CERTIFICATE

This is to certify that this manual is a bonafide record of practical work in the **DBMS Lab** in **Vth Semester of III Year ICD (CSE) program** during the academic year **2024-25**. This content was prepared by **Dr. Jagdeep Singh (Assistant Professor),** Department of Computer Science and Engineering.

Dr. Jagdeep Singh

AP, CSE,

SLIET LONGOWAL

Syllabus

Title of the course : **Database Management Lab**

Subject Code : **CS-313**

Course outcomes: At the end of the course, students will be able to:

CO1	Devise queries using DDL, DML, DCL and TCL commands.
CO2	Develop application programs using PL/SQL.
CO3	Create views, forms and reports.
CO4	Familiarization with different types of keys.

CO/PO Mapping : (Strong(S)/Medium(M)/Weak(W) indicates strength of correlation)										
COs	Programme Outcomes (POs)									
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10
CO1		S	S	M						S
CO2		S	M	S						S
CO3		S	S	M						S
CO4		S	M	M						S

LIST OF PRACTICALS

1. Introduction to various constraints such as Primary key, secondary key, Super key, etc.
2. To study Data Definition Commands (create, drop).
3. To study Data Manipulation Commands (insert, delete, update, select)
4. To study Data Control Commands (Commit, revoke, rollback, connect, execute)
5. Create Table, SQL for Insertion, Deletion, Update and Retrieval using aggregating functions.
6. Write Programs in PL/SQL, Understanding the concept of Cursors.
7. Write Program for Join, Union & intersection etc.
8. Creating Views, Writing Assertions, and Triggers.
9. Creating Forms, Reports etc.
10. WAP in PL/SQL for adding two numbers.
11. WAP in PL/SQL for reversing the number. For example the number is 12345 and reverse number will be 54321.
12. WAP in PL/SQL to find the number is even or odd.
13. WAP in PL/SQL to count numbers from 1 to 100.

PREFACE

This "**DBMS**" lab manual is designed to teach the fundamentals of data base design, emphasizing implementation in the SQL/PLSQL programming language. Readers of this manual are expected to be familiar with the syntax of SQL and similar procedural languages. DBMS concepts are increasingly critical to the IT industry, particularly for software development at the system level.

This practical manual has been carefully prepared to enhance the development of procedural programming skills. It includes a variety of exercises and their solutions so that students can understand them quickly and easily. This manual will prove valuable to Computer Science & Engineering students in grasping the applied aspects of database. There is always room for improvement, and we welcome suggestions from readers and users for future editions.

BY

Dr. Jagdeep Singh

AP, CSE, SLIET LONGOWAL

ACKNOWLEDGEMENT

It was a wonderful experience working on the “DBMS Lab” manual. First, I would like to express my sincere gratitude to **Prof. Birmohan Singh**, Head of the Department of Computer Science and Engineering, for his continuous support and technical guidance in preparing this document. I am deeply indebted and would like to acknowledge the invaluable support and patronage of **Prof. Mani Kant Paswan**, Director of the institute, for providing me with this excellent opportunity and his constant encouragement throughout the process. Finally, I extend my heartfelt thanks to the entire faculty of the CSE Department, whose inspiration and assistance helped me achieve this goal.

BY

Dr. Jagdeep Singh

AP, CSE, SLIET LONGOWAL

General Instructions

1. **Punctuality**: Students must arrive on time for the DBMS lab. Latecomers will not be allowed to participate in the lab session.
2. **Attendance**: Programs missed due to tardiness will be avoided. Students are expected to be on time.
3. **Preparation**: Students should prepare at home for the sessions' scheduled Programs.
4. **ID Cards**: Displaying an identity card is mandatory for entry into the lab.
5. **Mobile Phones**: Students need help to bring mobile phones into the lab.
6. **Responsibility for Equipment**: Any damage or loss of equipment, such as keyboards or mice, during the lab session will be the student's responsibility. A penalty or fine will be imposed if necessary.
7. **Lab Records**: Students must update their lab observation books and records after each session. Before leaving, they must get their lab observation book signed by the faculty member.
8. **Submission of Lab Records**: Lab records must be submitted to the faculty in the staffroom during the next lab session for correction and return.
9. **Movement in the Lab**: Students should remain at their assigned stations and avoid moving around during the lab session.
10. **Emergencies**: In an emergency, students must obtain written permission from the faculty member in charge.
11. **Disciplinary Action**: Faculty members can suspend students from the lab session for disciplinary reasons.
12. **Original Work**: Students should not copy outputs from others. They must write their own results.

BY

Dr. Jagdeep Singh

LIST OF PROGRAMS

1. Introduction to various constraints such as Primary key, secondary key, Super key, etc.
2. To study Data Definition Commands (create, drop).
3. To study Data Manipulation Commands (insert, delete, update, select)
4. To study Data Control Commands (Commit, revoke, rollback, connect, execute)
5. WAP in SQL for learning the concept VIEWS
6. WAP in SQL to understand the concept of MIN, MAX, AVG and COUNT, etc.
7. WAP in SQL using ORDER BY and GROUP BY clause.
8. WAP in SQL to understand the concept of subqueries.
9. WAP in SQL using FOREIGN KEY.
10. WAP in SQL to learn the concept of RANK() and DENSERANK().
11. WAP in SQL to CONCAT Function.
12. WAP in SQL to learn the concept for STRING FUNCTION.
13. WAP in SQL for CONDITIONAL statements
14. WAP in SQL to learn TRANSCATION.
15. WAP in SQL to learn the concept of TRIGGER .
16. WAP in SQL using UNION and INTERSECT
17. WAP in SQL for Join, Union & intersection etc.
18. WAP in PL/SQL for adding two numbers.
19. WAP in PL/SQL for reversing the number. For example the number is 12345 and reverse number will be 54321.
20. WAP in PL/SQL to find the number is even or odd.
21. WAP in PL/SQL to count numbers from 1 to 20.

Program 1. Introduction to various constraints such as Primary key, secondary key, Super key, etc.

1. Primary Key

- A unique identifier for each record in a table.
- **Characteristics**:
  - Must contain unique values.
  - Cannot contain NULL values.
  - There can only be one primary key per table, which may consist of a single or multiple columns (composite key).

2. Secondary Key

- An attribute or a set of attributes used to access records in a database but does not uniquely identify them.
- **Characteristics**:
  - Allows for efficient retrieval of records based on non-unique attributes.
  - Can contain duplicate values and NULLs.

3. Super Key

- A set of one or more attributes that can uniquely identify a record in a table.
- **Characteristics**:
  - Can include primary keys and any additional attributes.
  - Not necessarily minimal; a super key may contain extra attributes beyond those needed for uniqueness.

4. Candidate Key

- A minimal super key, meaning it uniquely identifies a record without any extra attributes.
- **Characteristics**:
  - A table can have multiple candidate keys.
  - One candidate key is chosen as the primary key.

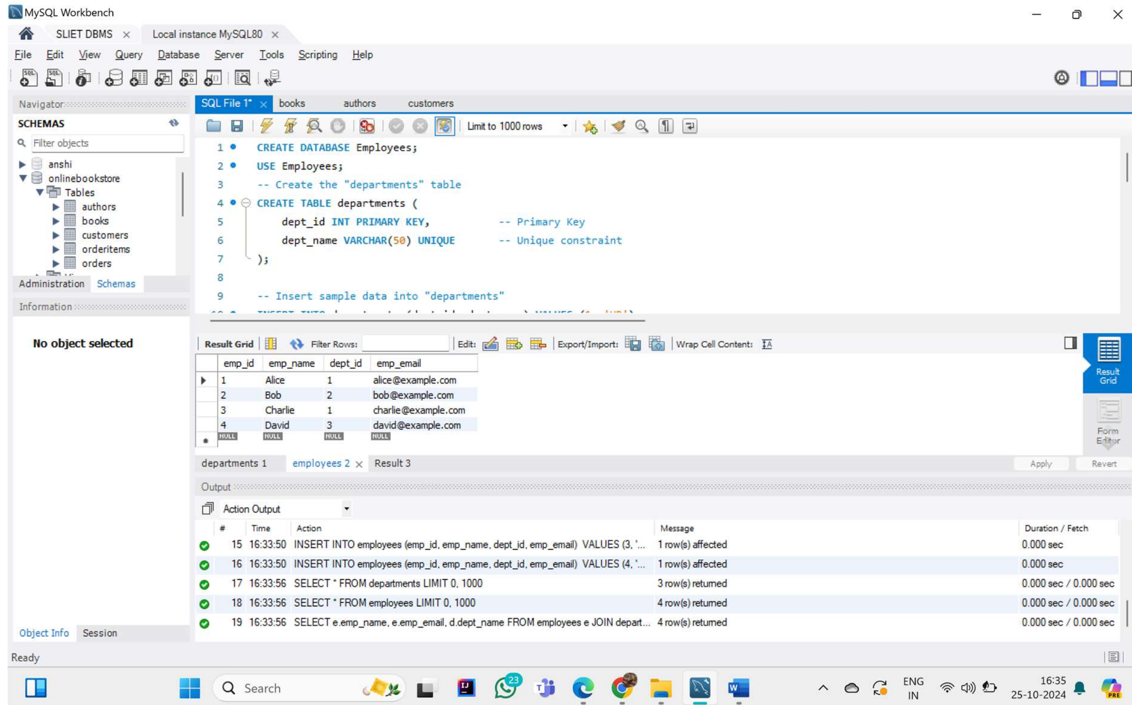
5. Foreign Key

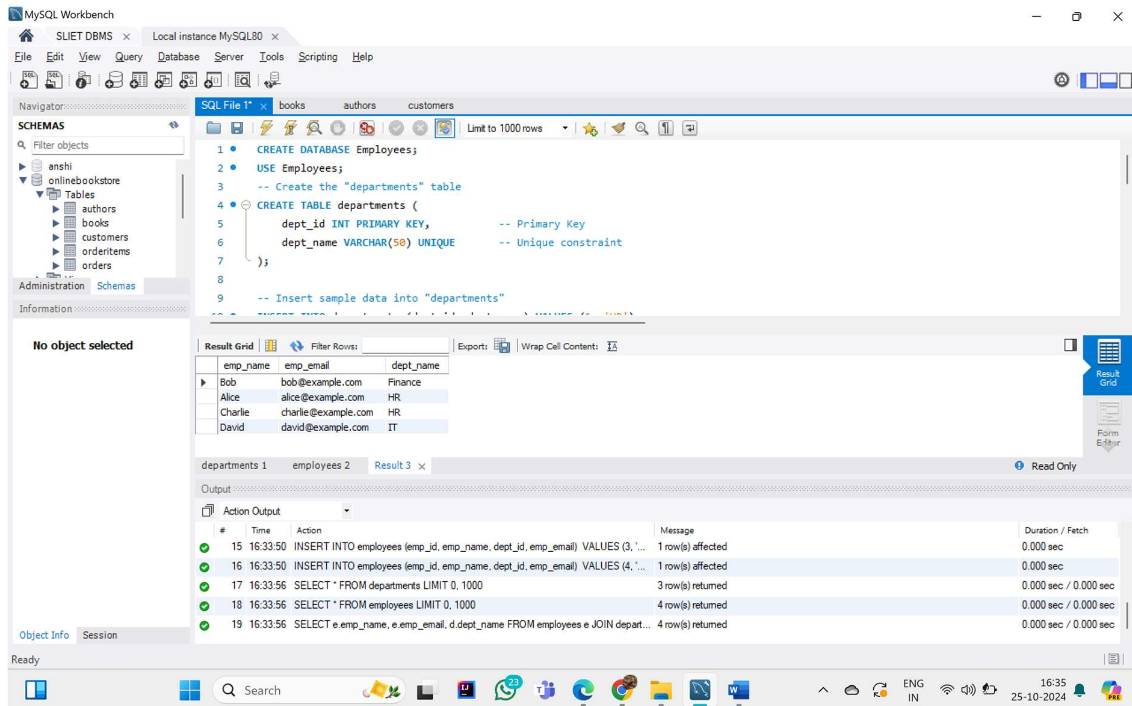
- An attribute or a set of attributes in one table that refers to the primary key in another table.
- **Characteristics**:
  - Establishes a relationship between two tables.
  - Can contain duplicate values and NULLs.

```
CREATE DATABASE Employees;
USE Employees;

-- Create the "departments" table
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,          -- Primary Key
    dept_name VARCHAR(50) UNIQUE      -- Unique constraint
);

-- Insert sample data into "departments"
INSERT INTO departments (dept_id, dept_name) VALUES (1, 'HR');
INSERT INTO departments (dept_id, dept_name) VALUES (2, 'Finance');
INSERT INTO departments (dept_id, dept_name) VALUES (3, 'IT');

-- Create the "employees" table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,           -- Primary Key
    emp_name VARCHAR(50) NOT NULL,    -- Not Null constraint
    dept_id INT,                      -- Foreign Key constraint
    emp_email VARCHAR(100) UNIQUE,    -- Unique constraint

    -- Define a foreign key constraint referencing "departments" table
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);

-- Insert sample data into "employees"
INSERT INTO employees (emp_id, emp_name, dept_id, emp_email)
VALUES (1, 'Alice', 1, 'alice@example.com');
INSERT INTO employees (emp_id, emp_name, dept_id, emp_email)
VALUES (2, 'Bob', 2, 'bob@example.com');
INSERT INTO employees (emp_id, emp_name, dept_id, emp_email)
```

```
VALUES (3, 'Charlie', 1, 'charlie@example.com');
INSERT INTO employees (emp_id, emp_name, dept_id, emp_email)
VALUES (4, 'David', 3, 'david@example.com');

-- Select query to display all records in the "departments" table
SELECT * FROM departments;

-- Select query to display all records in the "employees" table
SELECT * FROM employees;

-- Select query to retrieve employee names along with their department names
SELECT e.emp_name, e.emp_email, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

Output:


MySQL Workbench
SLIET DBMS
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
anshi
onlinebookstore
Tables
authors
books
customers
orderitems
orders
Administration Schemas
Information
No object selected
Object Info Session
SQL File 1* books authors customers
Limit to 1000 rows
1 CREATE DATABASE Employees;
2 USE Employees;
3 -- Create the "departments" table
4 CREATE TABLE departments (
5 dept_id INT PRIMARY KEY, -- Primary Key
6 dept_name VARCHAR(50) UNIQUE -- Unique constraint
7);
8
9 -- Insert sample data into "departments"
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:
emp_id emp_name dept_id emp_email
1 Alice 1 alice@example.com
2 Bob 2 bob@example.com
3 Charlie 1 charlie@example.com
4 David 3 david@example.com
NULL NULL NULL NULL
Result Grid
Form Editor
departments 1 employees 2 Result 3
Apply Revert
Output
Action Output
Time Action Message Duration / Fetch
15 16:33:50 INSERT INTO employees (emp_id, emp_name, dept_id, emp_email) VALUES (3, '... 1 row(s) affected 0.000 sec
16 16:33:50 INSERT INTO employees (emp_id, emp_name, dept_id, emp_email) VALUES (4, '... 1 row(s) affected 0.000 sec
17 16:33:56 SELECT * FROM departments LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec
18 16:33:56 SELECT * FROM employees LIMIT 0, 1000 4 row(s) returned 0.000 sec / 0.000 sec
19 16:33:56 SELECT e.emp_name, e.emp_email, d.dept_name FROM employees e JOIN depart... 4 row(s) returned 0.000 sec / 0.000 sec
Ready
Search
ENG IN
16:35
25-10-2024


MySQL Workbench
SLIET DBMS
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
anshi
onlinebookstore
Tables
authors
books
customers
orderitems
orders
Administration Schemas
Information
No object selected
Object Info Session
SQL File 1* books authors customers
Limit to 1000 rows
1 CREATE DATABASE Employees;
2 USE Employees;
3 -- Create the "departments" table
4 CREATE TABLE departments (
5 dept_id INT PRIMARY KEY, -- Primary Key
6 dept_name VARCHAR(50) UNIQUE -- Unique constraint
7);
8
9 -- Insert sample data into "departments"
Result Grid Filter Rows: Export: Wrap Cell Content:
emp_name emp_email dept_name
Bob bob@example.com Finance
Alice alice@example.com HR
Charlie charlie@example.com HR
David david@example.com IT
Result Grid
Form Editor
departments 1 employees 2 Result 3
Read Only
Output
Action Output
Time Action Message Duration / Fetch
15 16:33:50 INSERT INTO employees (emp_id, emp_name, dept_id, emp_email) VALUES (3, '... 1 row(s) affected 0.000 sec
16 16:33:50 INSERT INTO employees (emp_id, emp_name, dept_id, emp_email) VALUES (4, '... 1 row(s) affected 0.000 sec
17 16:33:56 SELECT * FROM departments LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec
18 16:33:56 SELECT * FROM employees LIMIT 0, 1000 4 row(s) returned 0.000 sec / 0.000 sec
19 16:33:56 SELECT e.emp_name, e.emp_email, d.dept_name FROM employees e JOIN depart... 4 row(s) returned 0.000 sec / 0.000 sec
Ready
Search
ENG IN
16:35
25-10-2024

Program 2. To study Data Definition Commands (create, drop).

CREATE Command: Used to create new database objects such as tables, views, indexes, or schemas.

```
CREATE TABLE Employees (

    EmployeeID INT PRIMARY KEY,

    FirstName VARCHAR(50),

    LastName VARCHAR(50),

    HireDate DATE

);
```

DROP Command: Used to delete existing database objects like tables, views, or indexes.

```
DROP TABLE Employees;
```

Output:

MySQL Workbench

```
9        HireDate DATE
10   );
11
12   -- Insert values into the Employees table
13   INSERT INTO Employees (EmployeeID, FirstName, LastName, HireDate)
14   VALUES
15       (1, 'John', 'Doe', '2022-01-15'),
16       (2, 'Jane', 'Smith', '2021-03-10'),
17       (3, 'Robert', 'Johnson', '2023-07-05');
18
```

EmployeeID	FirstName	LastName	HireDate
1	John	Doe	2022-01-15
2	Jane	Smith	2021-03-10
3	Robert	Johnson	2023-07-05
NULL	NULL	NULL	NULL

#	Time	Action	Message	Duration / Fetch
3	19:08:47	USE Sliet_emp	0 row(s) affected	0.000 sec
4	19:08:52	SELECT * FROM Employees LIMIT 0, 1000	Error Code: 1146. Table 'sliet_emp.employees' doesn't exist	0.000 sec
5	19:08:58	CREATE TABLE Employees (EmployeeID INT PRIMARY KEY, FirstName VA...	0 row(s) affected	0.015 sec
6	19:09:04	INSERT INTO Employees (EmployeeID, FirstName, LastName, HireDate) VALUES ...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.015 sec
7	19:09:08	SELECT * FROM Employees LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Query Completed

Program 3. To study Data Manipulation Commands (insert, delete, update, select)

INSERT Command: Adds new records (rows) to a table.

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, HireDate)

VALUES (1, 'John', 'Doe', '2023-01-15');
```

DELETE Command: Removes existing records from a table.

```
DELETE FROM Employees

WHERE EmployeeID = 1;
```

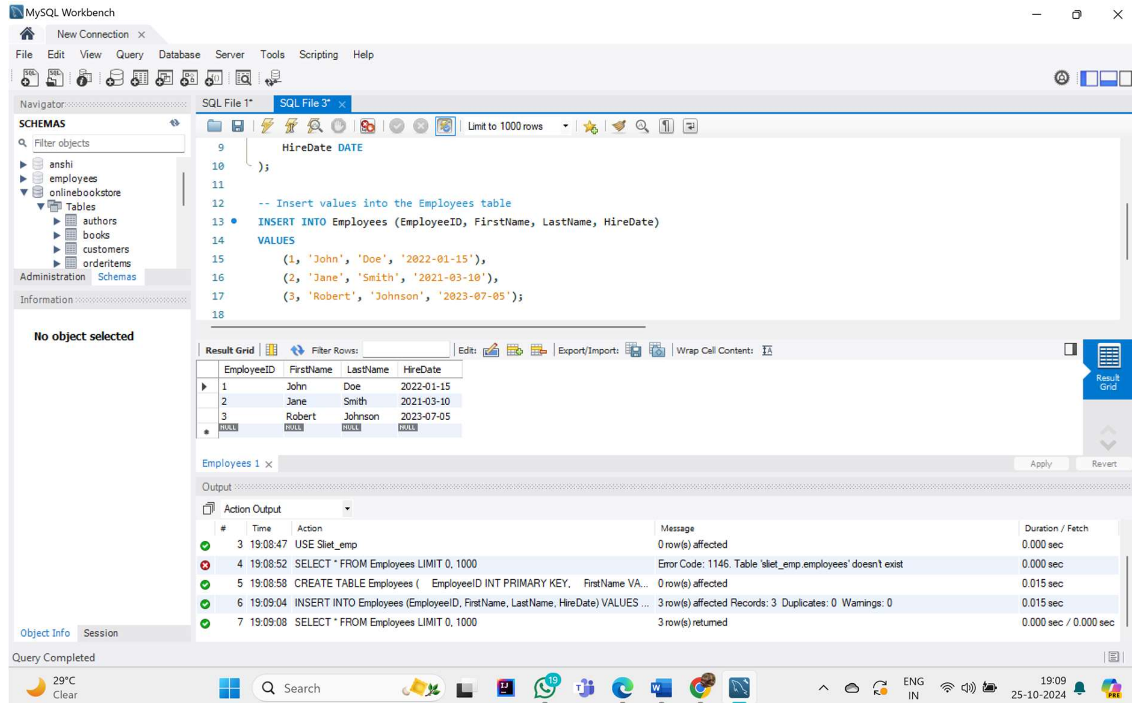
UPDATE Command: Modifies existing records in a table.

```
UPDATE Employees

SET LastName = 'Smith'

WHERE EmployeeID = 1;
```

SELECT Command: Retrieves data from one or more tables.

```
SELECT FirstName, LastName

FROM Employees

WHERE HireDate > '2022-01-01';
```

Output:



Programs 4

To study data control commands commit, revoke, rollback, connect , execute

COMMIT: Saves all changes made during the current transaction permanently in the database. Once committed, the changes cannot be rolled back. Syntax: COMMIT; REVOKE: Removes previously granted permissions or privileges from a user or role, which affects access to database objects like tables, views, or procedures. Syntax: REVOKE privilege_type ON object_name FROM user_name; ROLLBACK: Reverts all changes made during the current transaction to the last committed state, undoing any modifications since the transaction began. Syntax: ROLLBACK; CONNECT: Establishes a connection between the user and the database, allowing them to perform operations within that session. Syntax: CONNECT user_name/password@database_name;

EXECUTE: Runs a specified SQL command or stored procedure, often used in procedural languages like PL/SQL to execute code blocks, queries, or stored functions.

Syntax: EXECUTE procedure_name;

Programs 5

WAP in SQL for learning the concept VIEWS

A view in SQL is a virtual table that is based on the result of a query. It does not store data itself; instead, it provides a way to present data from one or more tables in a specific format or structure.

Virtual Table: A view is treated like a table in SQL, but it does not hold any data. Instead, it generates data dynamically when queried.

Views are a powerful feature in SQL that enhance data management, security, and usability. They allow you to encapsulate complex queries and present data in a user-friendly manner, making them valuable tools for database design and interaction.

```
CREATE DATABASE CompanyDB;

USE CompanyDB;

CREATE TABLE Employees (

  EmployeeID INT PRIMARY KEY,

 FirstName VARCHAR(50),

 LastName VARCHAR(50),

 Department VARCHAR(50),

 Salary DECIMAL(10, 2) );

INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)
VALUES

 (1, 'John', 'Doe', 'Sales', 60000.00),

(2, 'Jane', 'Smith', 'HR', 50000.00),

(3, 'Jim', 'Brown', 'Sales', 55000.00),

(4, 'Jake', 'White', 'IT', 70000.00);

SELECT * FROM Employees;
```

EmployeeID	FirstName	LastName	Department	Salary
1	John	Doe	Sales	60000.00
2	Jane	Smith	HR	50000.00
3	Jim	Brown	Sales	55000.00
4	Jake	White	IT	70000.00
NULL	NULL	NULL	NULL	NULL

```
CREATE VIEW SalesEmployees
```

```
AS SELECT EmployeeID, FirstName, LastName, Salary FROM Employees

WHERE Department = 'Sales';

SELECT * FROM SalesEmployees;
```

EmployeeID	FirstName	LastName	Salary
1	John	Doe	60000.00
3	Jim	Brown	55000.00

Programs 6

WAP in SQL to understand the concept of MIN,MAX,AVG and COUNT,etc.

The MIN function is used to retrieve the smallest value from a specified column in a table. It can be applied to numeric, date, or string columns.

The MAX function is used to find the largest value in a specified column. Similar to MIN, it can be applied to numeric, date, or string columns.

The AVG function calculates the average (mean) value of a numeric column. It sums all the values in the column and divides by the count of non-null entries.

The COUNT function counts the number of rows that meet a specified condition. It can be used to count all rows, non-null values in a column, or distinct values.

```
Create Database Employee; use Employee; create table employees( employee_id int primary key,

first_name varchar(20), last_name varchar(20), salary decimal(10,2), department varchar(10)

);

insert into employees(employee_id,first_name,last_name,salary,department) values

(1,"Priyanshu","Raj",90000,"Marketing"),

(2,"Sanskar","Raj",100000,"HR"),

(3,"Avinash","Aryan",900000,"Finance"),
```

(4,"Sanjeev","Kumar",900000,"IT"),

(5,"Krish","Rajan",100000,"Marketing"),

(6,"Aman","Raj",80000,"HR"),

(7,"Rahul","Raj",70000,"Finance"),

(8,"Alok","Raj",60000,"IT"); select * from

employees; select * from employees where

first_name like 'P%';

employee_id	first_name	last_name	salary	department
1	Priyanshu	Raj	90000.00	Marketing
2	Sanskar	Raj	100000.00	HR
3	Avinash	Aryan	900000.00	Finance
4	Sanjeev	Kumar	900000.00	IT
5	Krish	Rajan	100000.00	Marketing
6	Aman	Raj	80000.00	HR
7	Rahul	Raj	70000.00	Finance
8	Alok	Raj	60000.00	IT

select * from employees where salary IS NULL;

select * from employees where salary IS NOT NULL;

select count(*) as total_employees from employees;

total_employees
16

Select MAX(salary) as highest_salary from employees;

highest_salary
900000.00

Select MIN(salary) as lowest_salary from employees;

lowest_salary
10000.00

Select AVG(salary) as average_salary from employees;

	average_salary
▶	175000.000000

Select * from employees where salary > (select avg(salary) from employees);

	employee_id	first_name	last_name	salary	department
*	NULL	NULL	NULL	NULL	NULL

Programs 7

WAP in SQL using ORDER BY and GROUP BY clause.

ORDER BY
Purpose: To sort the result set of a query based on one or more columns.
Usage: It can sort data in ascending (ASC) or descending (DESC) order.

GROUP BY
Purpose: To group rows that have the same values in specified columns into summary rows, often used with aggregate functions (like COUNT, SUM, AVG).
Usage: It allows you to perform operations on each group of data.

```
CREATE DATABASE college;

USE College;

CREATE TABLE student(

roll_no INT PRIMARY KEY,

name VARCHAR(50),

marks FLOAT,

grade VARCHAR(5),

city VARCHAR(50)

);

INSERT INTO student(roll_no,name,marks,grade,city)

VALUES

(101,"Ram",97,"A","Delhi"),

(102,"Shyam",84,"B","Mumbai"),
```

```
(103,"Rohan",79,"C","Chennai"),
(104,"Narayan",88,"B","Bangaluru"),
(105,"Rakesh",63,"D","Kolkata");
SELECT DISTINCT grade FROM student;
```

grade
A
B
C
D

```
SELECT * FROM Student
WHERE marks+3=100;
```

roll_no	name	marks	grade	city
101	Ram	97	A	Delhi
NULL	NULL	NULL	NULL	NULL

```
SELECT * FROM Student
WHERE marks>80 OR city="Kolkata";
```

roll_no	name	marks	grade	city
101	Ram	97	A	Delhi
102	Shyam	84	B	Mumbai
104	Narayan	88	B	Bangaluru
105	Rakesh	63	D	Kolkata
NULL	NULL	NULL	NULL	NULL

```
SELECT * FROM Student
ORDER BY city ASC;
```

roll_no	name	marks	grade	city
104	Narayan	88	B	Bangaluru
103	Rohan	79	C	Chennai
101	Ram	97	A	Delhi
105	Rakesh	63	D	Kolkata
102	Shyam	84	B	Mumbai
NULL	NULL	NULL	NULL	NULL

```
SELECT * FROM Student
ORDER BY marks DESC
LIMIT 3;
```

roll_no	name	marks	grade	city
101	Ram	97	A	Delhi
104	Narayan	88	B	Bangaluru
102	Shyam	84	B	Mumbai
NULL	NULL	NULL	NULL	NULL

```
SELECT city,count(name)
FROM college.Student
Group By city;
```

city	count(name)
Delhi	1
Mumbai	1
Chennai	1
Bangaluru	1
Kolkata	1

```
SELECT city, avg(marks)
FROM student
GROUP BY city
ORDER BY city;
```

city	avg(marks)
Bangaluru	88
Chennai	79
Delhi	97
Kolkata	63
Mumbai	84

Program 8

WAP in SQL to understand the concept of subqueries.

A subquery, or inner query, is a query nested inside another SQL query (the outer query). Subqueries can be used in various parts of a SQL statement, including the SELECT, FROM, and WHERE clauses.

```
CREATE DATABASE retail_store;

USE retail_store;

CREATE TABLE employees (

    employee_id INT PRIMARY KEY,

    name VARCHAR(50),

    position VARCHAR(50),

    salary DECIMAL(10, 2)

);

CREATE TABLE customers (

    customer_id INT PRIMARY KEY,

    name VARCHAR(50),

    email VARCHAR(100)

);

INSERT INTO employees (employee_id, name, position, salary) VALUES

(1, 'Alice', 'Manager', 60000),

(2, 'Bob', 'Sales', 50000),

(3, 'Charlie', 'Sales', 50000),

(4, 'David', 'Support', 40000);

select*from employees;
```

	employee_id	name	position	salary
▶	1	Alice	Manager	60000.00
	2	Bob	Sales	50000.00
	3	Charlie	Sales	50000.00
	4	David	Support	40000.00
*	NULL	NULL	NULL	NULL

```
INSERT INTO customers (customer_id, name, email) VALUES
(1, 'John Doe', 'john@example.com'),
(2, 'Jane Smith', 'jane@example.com');
select*from customers;
```

customer_id	name	email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com
NULL	NULL	NULL

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

name	salary
Alice	60000.00

```
SELECT name
FROM employees
WHERE employee_id IN (SELECT customer_id FROM customers WHERE position = 'sales');
```

name
Bob

Program 9

WAP in SQL using FOREIGN KEY.

Referential integrity is a database concept that ensures relationships between tables remain consistent. It guarantees that a foreign key in one table must correspond to an existing primary key in another table. This helps maintain data accuracy and integrity across related tables.

Foreign Key Constraints: Referential integrity is enforced using foreign keys. A foreign key in a child table points to a primary key in a parent table, ensuring that any value in the foreign key column matches a value in the primary key column.

```
CREATE DATABASE School;

USE School;

-- Create Students table

CREATE TABLE Students (

    StudentID INT PRIMARY KEY ,

    FirstName VARCHAR(50),

    LastName VARCHAR(50)

);


-- Create Courses table

CREATE TABLE Courses (

    CourseID INT PRIMARY KEY,

    CourseName VARCHAR(100)

);


-- Create Enrollments table with foreign keys

CREATE TABLE Enrollments (

    EnrollmentID INT PRIMARY KEY,

    StudentID INT,

    CourseID INT,

    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

-- Insert sample data into Students

INSERT INTO Students (FirstName, LastName) VALUES ('John', 'Doe');

INSERT INTO Students (FirstName, LastName) VALUES ('Jane', 'Smith');

Select*from Students;
```

	StudentID	FirstName	LastName
▶	1	John	Doe
	2	Jane	Smith
*	NULL	NULL	NULL

```
-- Insert sample data into Courses
INSERT INTO Courses (CourseName) VALUES ('Mathematics');
INSERT INTO Courses (CourseName) VALUES ('Science');
Select*from Courses;
```

	CourseID	CourseName
▶	1	Mathematics
	2	Science
*	NULL	NULL

```
-- Insert sample data into Enrollments
INSERT INTO Enrollments (StudentID, CourseID) VALUES (1, 1);
INSERT INTO Enrollments (StudentID, CourseID) VALUES (2, 2);
SELECT
    s.FirstName,
    s.LastName,
    c.CourseName
FROM
    Enrollments e
JOIN
    Students s ON e.StudentID = s.StudentID
JOIN
    Courses c ON e.CourseID = c.CourseID;
```

	FirstName	LastName	CourseName
▶	John	Doe	Mathematics
	Jane	Smith	Science

Programs 10.

WAP in SQL to learn the concept of RANK() and DENSERANK().

RANK: This function assigns a rank to each row within a partition of a result set. If two or more students have the same score, they will receive the same rank, and the next rank(s) will be skipped. For example, if two students are tied for rank 1, the next rank will be 3.

DENSE_RANK: Similar to RANK, but it does not skip ranks. If two students have the same score, they receive the same rank, but the next rank will be the immediate next number. For example, if two students are tied for rank 1, the next rank will be 2.

```
CREATE DATABASE Scho;
USE Scho;
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Score INT
);
INSERT INTO Students (FirstName, LastName, Score) VALUES
('Alice', 'Johnson', 85),
('Bob', 'Smith', 95),
('Charlie', 'Brown', 85),
('David', 'Wilson', 70),
('Eve', 'Davis', 95);
SELECT
    StudentID,
    FirstName,
    LastName,
    Score,
```

```
RANK() OVER (ORDER BY Score DESC) AS Ran
FROM
    Students;
```

StudentID	FirstName	LastName	Score	Ran
2	Bob	Smith	95	1
5	Eve	Davis	95	1
1	Alice	Johnson	85	3
3	Charlie	Brown	85	3
4	David	Wilson	70	5

```
SELECT
    StudentID,
    FirstName,
    LastName,
    Score,
    DENSE_RANK() OVER (ORDER BY Score DESC) AS DenseRank
FROM
    Students;
```

StudentID	FirstName	LastName	Score	DenseRank
2	Bob	Smith	95	1
5	Eve	Davis	95	1
1	Alice	Johnson	85	2
3	Charlie	Brown	85	2
4	David	Wilson	70	3

Programs 11

WAP in SQL to CONCAT Function.

The **CONCAT** function in SQL is used to combine two or more strings into a single string. It is useful for creating full names, concatenating address fields, or forming any composite string from multiple sources.

```
CREATE DATABASE S;
USE S;
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
INSERT INTO Students (FirstName, LastName) VALUES
('Alice', 'Johnson'),
('Bob', 'Smith'),
('Charlie', 'Brown');
select *from Students;
```

StudentID	FirstName	LastName
1	Alice	Johnson
2	Bob	Smith
3	Charlie	Brown
NULL	NULL	NULL

```
SELECT
    StudentID,
    CONCAT(FirstName, ' ', LastName) AS FullName
FROM
    Students;
```

StudentID	FullName
1	Alice Johnson
2	Bob Smith
3	Charlie Brown

Program 12

WAP in SQL to learn the concept for STRING FUNCTION.

```
CREATE DATABASE raj;
USE raj;
CREATE TABLE Students (
   StudentID INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);
INSERT INTO Students (FirstName, LastName) VALUES
('Alice', 'Johnson'),
('Bob', 'Smith'),
('Charlie', 'Brown'),
('David', 'Wilson');
SELECT
   UPPER(FirstName) AS UpperFirstName,
   UPPER(LastName) AS UpperLastName
FROM
   Students;
```

UpperFirstName	UpperLastName
ALICE	JOHNSON
BOB	SMITH
CHARLIE	BROWN
DAVID	WILSON

```
SELECT
   LOWER(FirstName) AS LowerFirstName,
   LOWER(LastName) AS LowerLastName
FROM
   Students;
```

LowerFirstName	LowerLastName
alice	johnson
bob	smith
charlie	brown
david	wilson

```
SELECT
    FirstName,
    LastName,
    LENGTH(FirstName) AS FirstNameLength,
    LENGTH(LastName) AS LastNameLength
FROM
    Students;
```

FirstName	LastName	FirstNameLength	LastNameLength
Alice	Johnson	5	7
Bob	Smith	3	5
Charlie	Brown	7	5
David	Wilson	5	6

```
SELECT
    SUBSTRING(FirstName, 1, 3) AS FirstThreeLetters
FROM
    Students;
```

FirstThreeLetters
Ali
Bob
Cha
Dav

Programs 13: WAP in SQL for CONDITIONAL statements

CASE Statement: This allows you to perform conditional logic within your SQL queries.

WHEN: Specifies the condition to evaluate.

THEN: Defines the result to return if the condition is true.

ELSE: Specifies a default result if none of the conditions are met.

```
CREATE DATABASE khushi;
USE khushi;
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Score INT
);
INSERT INTO Students (FirstName, LastName, Score) VALUES
('Alice', 'Johnson', 85),
('Bob', 'Smith', 92),
('Charlie', 'Brown', 75),
('David', 'Wilson', 88),
('Eve', 'Davis', 65);
select *from Students;
```

StudentID	FirstName	LastName	Score
1	Alice	Johnson	85
2	Bob	Smith	92
3	Charlie	Brown	75
4	David	Wilson	88
5	Eve	Davis	65
NULL	NULL	NULL	NULL

```
SELECT
    FirstName,
    LastName,
    Score,
    CASE
```

```
        WHEN Score >= 90 THEN 'A'
        WHEN Score >= 80 THEN 'B'
        WHEN Score >= 70 THEN 'C'
        WHEN Score >= 60 THEN 'D'
        ELSE 'F'
    END AS Grade
FROM
    Students;
```

FirstName	LastName	Score	Grade
Alice	Johnson	85	B
Bob	Smith	92	A
Charlie	Brown	75	C
David	Wilson	88	B
Eve	Davis	65	D

Program 14: WAP in SQL to learn TRANSCATION.

A transaction in SQL is a sequence of one or more SQL operations that are executed as a single unit of work. Transactions are crucial for maintaining data integrity and consistency in a database, especially when multiple operations need to be treated as a cohesive operation.

```
CREATE DATABASE BankDB;
USE BankDB;
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY AUTO_INCREMENT,
    AccountHolder VARCHAR(100),
    Balance DECIMAL(10, 2)
);
INSERT INTO Accounts (AccountHolder, Balance) VALUES ('Alice', 500.00);
INSERT INTO Accounts (AccountHolder, Balance) VALUES ('Bob', 300.00);
-- Start a transaction
START TRANSACTION;
-- Step 1: Deduct money from Account A (Alice)
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
```

```
-- Step 2: Add money to Account B (Bob)
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;
-- Check Alice's balance after deduction
SELECT Balance FROM Accounts WHERE AccountID = 1;
-- If everything looks good, then commit the transaction
COMMIT; -- Or ROLLBACK; if you see an issue
```

	Balance
▶	400.00

Program 15

WAP in SQL to learn the concept of TRIGGER .

A **trigger** in SQL is a special kind of stored procedure that automatically executes (or "fires") in response to specific events on a particular table or view. Triggers are typically used to enforce business rules, maintain data integrity, or automate actions within the database without requiring additional application code.

```
CREATE DATABASE C;
USE C;
-- Create Employees Table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Salary DECIMAL(10, 2)
);

-- Create a Log Table to capture changes
CREATE TABLE EmployeeLog (
    LogID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    EmployeeID INT,
    Action VARCHAR(50),
    LogTime DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
DELIMITER //

CREATE TRIGGER AfterEmployeeInsert
AFTER INSERT ON Employees
FOR EACH ROW
BEGIN
    INSERT INTO EmployeeLog (EmployeeID, Action)
    VALUES (NEW.EmployeeID, 'Inserted');
END //

CREATE TRIGGER AfterEmployeeUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
    INSERT INTO EmployeeLog (EmployeeID, Action)
    VALUES (NEW.EmployeeID, 'Updated');
END //

-- Insert an employee
INSERT INTO Employees (FirstName, LastName, Salary) VALUES ('John', 'Doe', 50000.00);

-- Update an employee's salary
UPDATE Employees SET Salary = 55000.00 WHERE EmployeeID = 1;

-- Check the logs
SELECT * FROM Employees;
```

	EmployeeID	FirstName	LastName	Salary
▶	1	John	Doe	55000.00
*	NULL	NULL	NULL	NULL

SELECT * FROM EmployeeLog;

	LogID	EmployeeID	Action	LogTime
▶	1	1	Inserted	2024-10-19 11:16:31
	2	1	Updated	2024-10-19 11:16:31
*	NULL	NULL	NULL	NULL

Program 16

WAP in SQL using UNION and INTERSECT

UNION

- **Purpose**: Combines the result sets of two or more SELECT statements into a single result set.
- **Duplicates**: By default, UNION removes duplicate rows from the result set. If you want to include duplicates, you can use UNION ALL.
- **Requirements**: Each SELECT statement must have the same number of columns in the result set, and the columns must have compatible data types.

-> **INTERSECT**

- **Purpose**: Returns only the rows that appear in both result sets from two SELECT statements.
- **Duplicates**: INTERSECT automatically removes duplicates, returning only unique rows that are common to both sets.
- **Requirements**: Similar to UNION, each SELECT statement must have the same number of columns and compatible data types.

```
-- Create a Database
CREATE DATABASE SampleDB;
USE SampleDB;

-- Create First Table: Employees
CREATE TABLE Employees (
```

```
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

-- Create Second Table: Managers
CREATE TABLE Managers (
    ManagerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

-- Insert Sample Data into Employees
INSERT INTO Employees (FirstName, LastName) VALUES ('Alice', 'Smith');
INSERT INTO Employees (FirstName, LastName) VALUES ('Bob', 'Johnson');
INSERT INTO Employees (FirstName, LastName) VALUES ('Charlie', 'Brown');

-- Insert Sample Data into Managers
INSERT INTO Managers (FirstName, LastName) VALUES ('Alice', 'Smith');
INSERT INTO Managers (FirstName, LastName) VALUES ('David', 'Wilson');
-- UNION Example
SELECT FirstName, LastName FROM Employees
UNION
SELECT FirstName, LastName FROM Managers;
```

FirstName	LastName
Alice	Smith
Bob	Johnson
Charlie	Brown
David	Wilson

```
-- INTERSECT Example
```

```
SELECT FirstName, LastName FROM Employees
INTERSECT
SELECT FirstName, LastName FROM Managers;
```

	FirstName	LastName
▶	Alice	Smith

Program 17. Write Program for Join, Union & intersection etc.

```
CREATE DATABASE IF NOT EXISTS Sliet_Cse;
USE Sliet_Cse;

CREATE TABLE Employee_1(
Department_Name VARCHAR(100) NOT NULL,
Department_Id INT NOT NULL,
Employee_Name VARCHAR(50) NOT NULL,
Employee_Id INT PRIMARY KEY
);

/*
Department Id for Various Departments:-
1. Computer Science Department :- 101
2. Mechanical Department       :- 102
3. Electrical Department       :- 103
4. Civil Department            :- 104
5. Electronics Department      :- 105

*/
INSERT INTO Employee_1
```

```
(Employee_Name,Department_Name,Employee_Id,Department_Id)
VALUES
("Ansh Girdher","Computer Science Department",2211086,101),
("Lavish Garg","Computer Science Department",2211094,101),
("keshav Khatak","Civil Department",2211097,104),
("Dinesh Kumar","Electronics Department",2211093,105),
("Anil Taak","Electronics Department",2211033,105),
("Manav Garg","Mechanical Department",2211096,102),
("Aditya Chauduary","Mechanical Department",2216551,102),
("Anmoldeep Singh","Electrical Department",2214126,103),
("Dipanshu","Electrical Department",2216553,103);

SELECT * FROM Employee_1 ORDER BY Department_Id ASC;

CREATE TABLE Seminar(
Department_Id INT NOT NULL,
Domain VARCHAR(100) NOT NULL
);
INSERT INTO Seminar
(Department_Id,Domain)
VALUES
(101,"Artificial Intelligence"),
(102,"AutoCAD"),
(103,"MATLAB"),
(104,"Google Sketchup"),
(105,"PIC Microcontroller");

SELECT * FROM Seminar ORDER BY Department_id ASC;

-- Inner Join
SELECT *
```

```
FROM Employee_1
INNER JOIN Seminar
ON Employee_1.Department_id = Seminar.Department_id;

-- Left Join
SELECT *
FROM Employee_1
LEFT JOIN Seminar
ON Employee_1.Department_id = Seminar.Department_id;

-- Right Join
SELECT *
FROM Employee_1
RIGHT JOIN Seminar
ON Employee_1.Department_id = Seminar.Department_id;

-- Full Join
SELECT * FROM Employee_1 as a
LEFT JOIN Seminar as b
ON a.Department_id = b.Department_id
UNION
SELECT * FROM Employee_1 as a
RIGHT JOIN Seminar as b
ON a.Department_id = b.Department_id
```

Output:

MySQL Workbench

```sql
-- Full Join
SELECT * FROM Employee_1 as a
LEFT JOIN Seminar as b
ON a.Department_id = b.Department_id
UNION
SELECT * FROM Employee_1 as a
RIGHT JOIN Seminar as b
ON a.Department_id = b.Department_id
```

Department_Name	Department_Id	Employee_Name	Employee_Id	Department_Id	Domain
Electronics Department	105	Anil Taak	2211033	105	PIC Microcontroller
Computer Science Department	101	Ansh Girdher	2211086	101	Artificial Intelligence
Electronics Department	105	Dinesh Kumar	2211093	105	PIC Microcontroller
Computer Science Department	101	Lavish Garg	2211094	101	Artificial Intelligence
Mechanical Department	102	Manav Garg	2211096	102	AutoCAD

Action Output

#	Time	Action	Message	Duration / Fetch
23	16:39:46	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.000 sec / 0.000 sec
24	16:40:23	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec
25	16:41:55	SELECT * FROM Employee_1 RIGHT JOIN Seminar ON Employee_1.Department_i...	9 row(s) returned	0.000 sec / 0.000 sec
26	16:42:59	SELECT * FROM Employee_1 as a LEFT JOIN Seminar as b ON a.Department_id ...	9 row(s) returned	0.015 sec / 0.000 sec
27	16:43:07	SELECT * FROM Employee_1 as a LEFT JOIN Seminar as b ON a.Department_id ...	9 row(s) returned	0.000 sec / 0.000 sec

Query Completed

MySQL Workbench

```sql
CREATE DATABASE IF NOT EXISTS Sliet_Cse;
USE Sliet_Cse;

CREATE TABLE Employee_1(
Department_Name VARCHAR(100) NOT NULL,
Department_Id INT NOT NULL,
Employee_Name VARCHAR(50) NOT NULL,
Employee_Id INT PRIMARY KEY
);
```

Department_Name	Department_Id	Employee_Name	Employee_Id	Department_Id	Domain
Electronics Department	105	Anil Taak	2211033	105	PIC Microcontroller
Computer Science Department	101	Ansh Girdher	2211086	101	Artificial Intelligence
Electronics Department	105	Dinesh Kumar	2211093	105	PIC Microcontroller
Computer Science Department	101	Lavish Garg	2211094	101	Artificial Intelligence
Mechanical Department	102	Manav Garg	2211096	102	AutoCAD

Action Output

#	Time	Action	Message	Duration / Fetch
19	16:37:06	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	Error Code: 1146. Table 'employees.employee_1' doesn't exist	0.000 sec
20	16:37:13	USE Sliet_Cse	0 row(s) affected	0.000 sec
21	16:37:22	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.047 sec / 0.000 sec
22	16:37:39	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec
23	16:39:46	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.000 sec / 0.000 sec

Query Completed

```sql
CREATE DATABASE IF NOT EXISTS Sliet_Cse;
USE Sliet_Cse;

CREATE TABLE Employee_1(
Department_Name VARCHAR(100) NOT NULL,
Department_Id INT NOT NULL,
Employee_Name VARCHAR(50) NOT NULL,
Employee_Id INT PRIMARY KEY
);
```

Department_Name	Department_Id	Employee_Name	Employee_Id	Department_Id	Domain
Electronics Department	105	Anil Taak	2211033	105	PIC Microcontroller
Computer Science Department	101	Ansh Girdher	2211086	101	Artificial Intelligence
Electronics Department	105	Dinesh Kumar	2211093	105	PIC Microcontroller
Computer Science Department	101	Lavish Garg	2211094	101	Artificial Intelligence
Mechanical Department	102	Manav Garg	2211096	102	AutoCAD

#	Time	Action	Message	Duration / Fetch
20	16:37:13	USE Sliet_Cse	0 row(s) affected	0.000 sec
21	16:37:22	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.047 sec / 0.000 sec
22	16:37:39	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec
23	16:39:46	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.000 sec / 0.000 sec
24	16:40:23	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec

```sql
-- Right Join
SELECT *
FROM Employee_1
RIGHT JOIN Seminar
ON Employee_1.Department_id = Seminar.Department_id;

-- Full Join
SELECT * FROM Employee_1 as a
LEFT JOIN Seminar as b
```

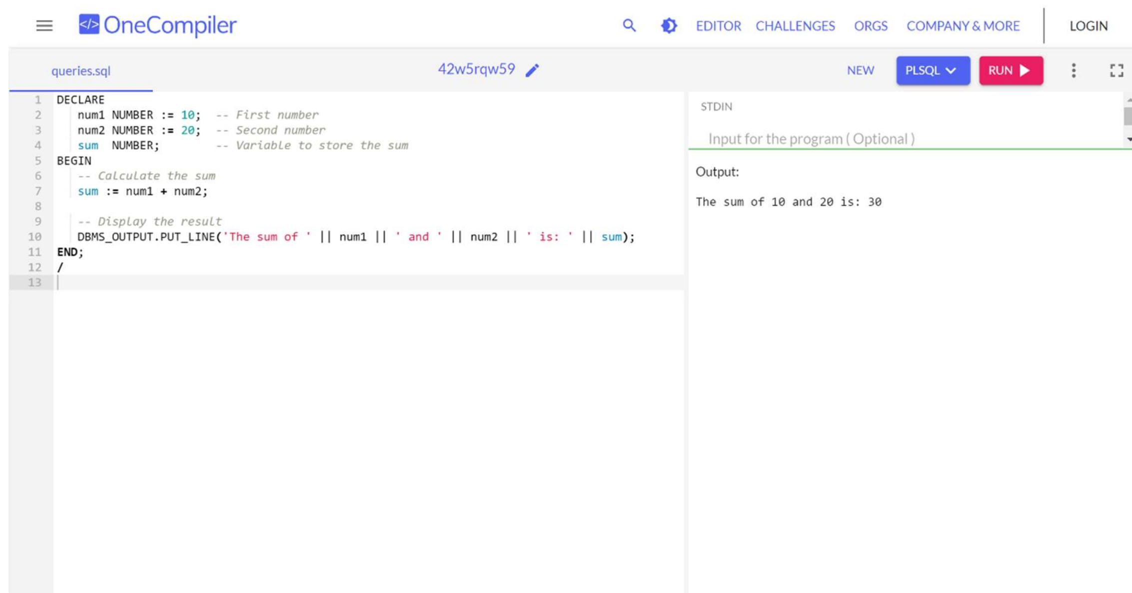
Department_Name	Department_Id	Employee_Name	Employee_Id	Department_Id	Domain
Computer Science Department	101	Lavish Garg	2211094	101	Artificial Intelligence
Computer Science Department	101	Ansh Girdher	2211086	101	Artificial Intelligence
Mechanical Department	102	Aditya Chauduary	2216551	102	AutoCAD
Mechanical Department	102	Manav Garg	2211096	102	AutoCAD
Electrical Department	103	Dipanshu	2216553	103	MATLAB

#	Time	Action	Message	Duration / Fetch
21	16:37:22	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.047 sec / 0.000 sec
22	16:37:39	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec
23	16:39:46	SELECT * FROM Employee_1 INNER JOIN Seminar ON Employee_1.Departmen...	9 row(s) returned	0.000 sec / 0.000 sec
24	16:40:23	SELECT * FROM Employee_1 LEFT JOIN Seminar ON Employee_1.Department_id...	9 row(s) returned	0.000 sec / 0.000 sec
25	16:41:55	SELECT * FROM Employee_1 RIGHT JOIN Seminar ON Employee_1.Department_i...	9 row(s) returned	0.000 sec / 0.000 sec

Program 18. WAP in PL/SQL for adding two numbers.

```
DECLARE
  num1 NUMBER := 10;  -- First number
  num2 NUMBER := 20;  -- Second number
  sum  NUMBER;        -- Variable to store the sum
BEGIN
  -- Calculate the sum
  sum := num1 + num2;

  -- Display the result
  DBMS_OUTPUT.PUT_LINE('The sum of ' || num1 || ' and ' || num2 || ' is: ' || sum);
END;
/
```

Output:



Program 19. WAP in PL/SQL for reversing the number. For example the number is 12345 and reverse number will be 54321.

```
DECLARE
   original_num NUMBER := 12345;  -- Original number to be reversed
   reversed_num NUMBER := 0;      -- Variable to store the reversed number
   remainder    NUMBER;           -- Variable to store the remainder
BEGIN
   -- Display the original number
   DBMS_OUTPUT.PUT_LINE('Original number: ' || original_num);

   -- Reverse the number
   WHILE original_num > 0 LOOP
      remainder := MOD(original_num, 10);              -- Get the last digit
      reversed_num := (reversed_num * 10) + remainder;  -- Append the last digit to the reversed
number
      original_num := TRUNC(original_num / 10);        -- Remove the last digit from the original
number
   END LOOP;

   -- Display the reversed number
   DBMS_OUTPUT.PUT_LINE('Reversed number: ' || reversed_num);
END;
/
```

OneCompiler EDITOR CHALLENGES ORGS COMPANY & MORE LOGIN

queries.sql 42w5rqw59 NEW PLSQL RUN

```
DECLARE
  original_num NUMBER := 12345;  -- Original number to be reversed
  reversed_num NUMBER := 0;      -- Variable to store the reversed number
  remainder    NUMBER;           -- Variable to store the remainder
BEGIN
  -- Display the original number
  DBMS_OUTPUT.PUT_LINE('Original number: ' || original_num);

  -- Reverse the number
  WHILE original_num > 0 LOOP
    remainder := MOD(original_num, 10);                -- Get the last digit
    reversed_num := (reversed_num * 10) + remainder;   -- Append the last digit to the rev
    original_num := TRUNC(original_num / 10);          -- Remove the last digit from the o
  END LOOP;

  -- Display the reversed number
  DBMS_OUTPUT.PUT_LINE('Reversed number: ' || reversed_num);
END;
/
```

STDIN ctrl + enter

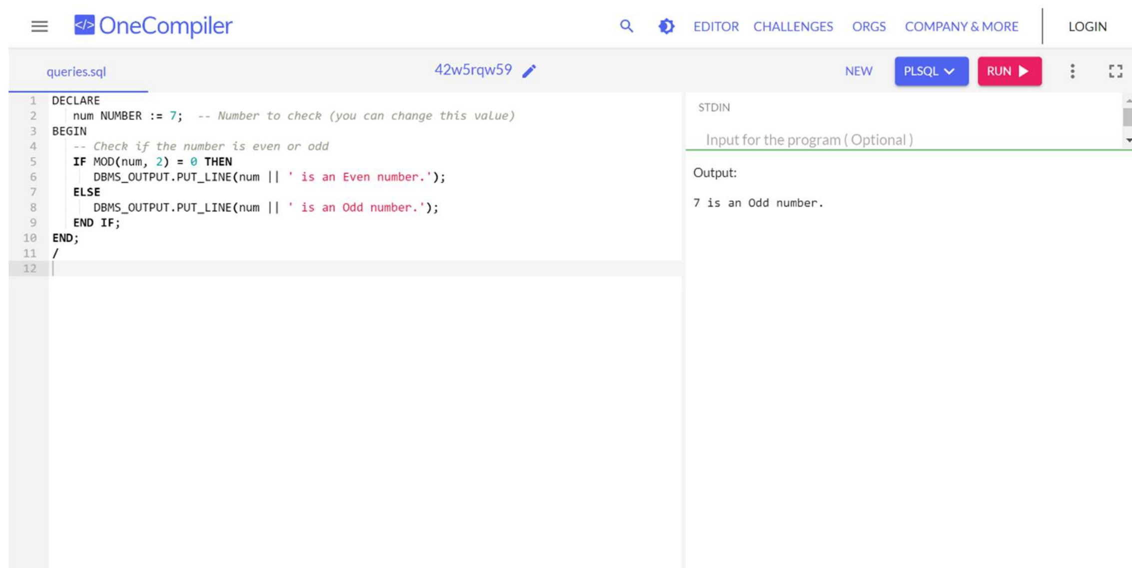
Input for the program (Optional)

Output:

```
Original number: 12345
Reversed number: 54321
```

Program 20. WAP in PL/SQL to find the number is even or odd.

```
DECLARE
  num NUMBER := 7;  -- Number to check (you can change this value)
BEGIN
  -- Check if the number is even or odd
  IF MOD(num, 2) = 0 THEN
    DBMS_OUTPUT.PUT_LINE(num || ' is an Even number.');
  ELSE
    DBMS_OUTPUT.PUT_LINE(num || ' is an Odd number.');
  END IF;
END;
/
```



Program 21. WAP in PL/SQL to count numbers from 1 to 20.

```
DECLARE
  counter NUMBER := 1;  -- Start counter from 1
BEGIN
  WHILE counter <= 20 LOOP
    DBMS_OUTPUT.PUT_LINE(counter);  -- Display the current number
    counter := counter + 1;         -- Increment the counter by 1
  END LOOP;
END;
/
```

Output

OneCompiler EDITOR CHALLENGES ORGS COMPANY & MORE LOGIN

queries.sql 42w5rqw59 NEW PLSQL RUN

```
DECLARE
  counter NUMBER := 1;  -- Start counter from 1
BEGIN
  WHILE counter <= 20 LOOP
    DBMS_OUTPUT.PUT_LINE(counter);  -- Display the current number
    counter := counter + 1;         -- Increment the counter by 1
  END LOOP;
END;
/
```

STDIN

Input for the program (Optional)

Output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```