Practical 1

CREATE TABLE workers ( Sr\_No INT,Name VARCHAR(50),Designation VARCHAR(50), Branch VARCHAR(50));

INSERT INTO workers (Sr\_No, Name, Designation, Branch) VALUES (1, 'Ram', 'Manager', 'Chennai'), (2, 'Santosh', 'Supervisor', 'Goa'),(3, 'Hari', 'Assistant', 'Delhi'), (4, 'Jagdesh', 'Peon', 'Mumbai');

**ALTER TABLE workers ADD Salary DECIMAL(10, 2)**; //This command modifies the workers table by adding a new column named **Salary**.

 The **Salary** column has a **DECIMAL(10, 2)** data type:

* 10 is the total number of digits the salary can have.
* 2 is the number of digits after the decimal point (precision of two decimal places).

**ALTER TABLE workers MODIFY Name VARCHAR(100);**

// This command removes all rows from the workers table.

 Unlike DELETE, TRUNCATE is a DML command that:

* Is faster than DELETE because it doesn't generate row-by-row logging.
* Cannot be rolled back in some databases (depending on the database system), and does not fire triggers.

 Important: The structure of the table remains intact, but all data is removed.

 After this command, the workers table will be empty, but its schema will remain**.**

**DESCRIBE workers;** // The DESCRIBE command provides information about the structure of the workers table.

It shows the columns, their data types, and any additional properties (like whether a column is NULL or has a DEFAULT value, etc.).

**CREATE TABLE emp AS SELECT \* FROM workers;** // This command creates a new table named emp that is a copy of the workers table.

 The SELECT \* FROM workers part selects all rows and columns from the workers table.

 The CREATE TABLE emp AS part creates the emp table and populates it with the data from workers.

 Essentially, emp will have the same structure and data as workers at this poin

TRUNCATE TABLE workers;

DELETE FROM workers WHERE Sr\_No = 2;

DROP TABLE workers;

Practical 2

-- Create Customers table

CREATE TABLE Customers ( cname VARCHAR(50) PRIMARY KEY, city VARCHAR(50)); //primary key means evry cust should have uniq name

-- Create Branch table

CREATE TABLE Branch (bname VARCHAR(50) PRIMARY KEY, city VARCHAR(50));

-- Create Deposit table with foreign keys referencing Customers and Branch

**CREATE TABLE Deposit ( actno INT PRIMARY KEY, cname VARCHAR(50),bname VARCHAR(50), amount DECIMAL(10, 2), adate DATE, FOREIGN KEY (cname) REFERENCES Customers(cname), FOREIGN KEY (bname) REFERENCES Branch(bname));**

// A DECIMAL(10, 2) column that stores the amount of the deposit. It can have up to 10 digits in total, with 2 digits after the decimal point. The FOREIGN KEY constraints ensure that the cname and bname in the Deposit table must match valid entries in the Customers and Branch tables, respectively.

-- Create Borrow table with foreign keys referencing Customers and Branch

**CREATE TABLE Borrow ( loanno INT PRIMARY KEY, cname VARCHAR(50),**

**bname VARCHAR(50),**

**amount DECIMAL(10, 2),**

**FOREIGN KEY (cname) REFERENCES Customers(cname),** //  FOREIGN KEY (cname) REFERENCES Customers(cname) ensures that the cname in Deposit or Borrow must exist in Customers.

 FOREIGN KEY (bname) REFERENCES Branch(bname) ensures that the bname in Deposit or Borrow must exist in Branch.

**FOREIGN KEY (bname) REFERENCES Branch(bname) );**

-- Insert data into Branch table

INSERT INTO Branch (bname, city) VALUES ('Main', 'CityA');

INSERT INTO Branch (bname, city) VALUES ('North', 'CityB');

INSERT INTO Branch (bname, city) VALUES ('South', 'CityC');

-- Insert data into Customers table

INSERT INTO Customers (cname, city) VALUES ('Anil', 'CityA');

INSERT INTO Customers (cname, city) VALUES ('Sunita', 'CityB');

INSERT INTO Customers (cname, city) VALUES ('Raj', 'CityC');

-- Insert data into Deposit table

INSERT INTO Deposit (actno, cname, bname, amount, adate) VALUES (1001, 'Anil', 'Main', 5000.00, '1997-01-10');

INSERT INTO Deposit (actno, cname, bname, amount, adate) VALUES (1002, 'Sunita', 'North', 3000.00, '1996-12-05');

INSERT INTO Deposit (actno, cname, bname, amount, adate) VALUES (1003, 'Raj', 'South', 4500.00, '1996-11-25');

-- Insert data into Borrow table

INSERT INTO Borrow (loanno, cname, bname, amount) VALUES (2001, 'Anil', 'Main', 2000.00);

INSERT INTO Borrow (loanno, cname, bname, amount) VALUES (2002, 'Sunita', 'North', 1500.00);

INSERT INTO Borrow (loanno, cname, bname, amount) VALUES (2003, 'Raj', 'South', 2500.00);

----Perform Queries

1. Display names of depositors having an amount greater than 4000.

SELECT cname

FROM Deposit

WHERE amount > 4000;

2. Display the account date of customer Anil.

SELECT adate

FROM Deposit

WHERE cname = 'Anil';

3. Display account number and deposit amount of customers with accounts opened between 1-12-96 and 1-5-97.

SELECT actno, amount

FROM Deposit

WHERE adate BETWEEN '1996-12-01' AND '1997-05-01';

4. Find the names of all branches where the average account balance is more than 1200.

**SELECT bname**

**FROM Deposit**

**GROUP BY bname**

**HAVING AVG(amount) > 1200;** // he GROUP BY clause is used to **group rows** that have the same values in the specified column(s) into summary rows, like "grouping by branch name" here.

 In this case, we are grouping all the records (rows) in the Deposit table by the branch name (bname). This means that for each branch, the query will calculate aggregate functions (such as AVG(amount) in this case) on the deposit amounts.

5. Delete depositors having deposits less than 5000.

DELETE FROM Deposit

WHERE amount < 5000;

6. Create a view on the Deposit table.

CREATE VIEW DepositView AS

SELECT actno, cname, bname, amount, adate

FROM Deposit;

Practical 3

-- Create the Company database

CREATE DATABASE Company;

-- Use the Company database

USE Company;

-- Create the dept table

CREATE TABLE dept (

deptno INT PRIMARY KEY,

deptname VARCHAR(50),

location VARCHAR(50)

);

-- Create the emp table with foreign key constraint

CREATE TABLE emp (

eno INT PRIMARY KEY,

ename VARCHAR(50),

job VARCHAR(50),

hiredate DATE,

salary DECIMAL(10, 2),

commission DECIMAL(10, 2),

deptno INT,

FOREIGN KEY (deptno) REFERENCES dept(deptno)

);

//eno: This column represents the employee number. It's of type INT (integer), meaning it will store numeric values. PRIMARY KEY: This constraint ensures that the values in the eno column are unique and not NULL. Every employee must have a unique employee number, and it will be used to uniquely identify each row in the table.

//A foreign key is a column (or a set of columns) in one table that refers to the primary key in another table. In this case, deptno in the emp table is a foreign key.

//REFERENCES dept(deptno): This part establishes a relationship between the emp table and the dept table:

The deptno column in the emp table must contain values that already exist in the deptno column of the dept table.

This ensures referential integrity, meaning each employee must belong to a valid department that exists in the dept table.

If a department with a specific deptno does not exist in the dept table, you cannot insert an employee record into the emp table with that deptno.

-- Insert data into dept table

INSERT INTO dept (deptno, deptname, location) VALUES (10, 'Sales', 'New York');

INSERT INTO dept (deptno, deptname, location) VALUES (20, 'Development', 'Los Angeles');

INSERT INTO dept (deptno, deptname, location) VALUES (30, 'Marketing', 'Chicago');

-- Insert data into emp table

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (1, 'Ian', 'Salesman', '1980-05-01', 3000.00, 500.00, 10);

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (2, 'Ivy', 'Developer', '1981-06-01', 3500.00, NULL, 20);

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (3, 'John', 'Manager', '1979-09-01', 5000.00, NULL, 30);

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (4, 'Alice', 'Developer', '1980-11-01', 2800.00, NULL, 20);

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (5, 'Irene', 'Salesman', '1982-01-01', 3200.00, 400.00, 10);

INSERT INTO emp (eno, ename, job, hiredate, salary, commission, deptno) VALUES (6, 'Bob', 'Salesman', '1980-08-15', 3400.00, 450.00, 10);

3. Perform Queries

1. List the maximum salary paid to salesmen.

SELECT MAX(salary) AS max\_salary

FROM emp

WHERE job = 'Salesman';

//The MAX() function is an **aggregate function** that returns the highest value in a given column.

2. List the name of employees whose name starts with 'I'.

SELECT ename

FROM emp

WHERE ename LIKE 'I%';

//The LIKE operator is used to search for a pattern. 'I%' means the name starts with 'I' and can have any characters after that.

* The % symbol is a wildcard that matches zero or more characters.

3. List details of employees who joined before '30-sept-81'.

SELECT \*

FROM emp

WHERE hiredate < '1981-09-30';

4. List the employee details in the order of their basic salary.

SELECT \*

FROM emp

ORDER BY salary; // **ORDER BY salary**: This orders the result by the salary column in **ascending** order by default (lowest salary first).

5. List the number of employees and average salary for employees in department number "20".

SELECT COUNT(\*) AS no\_of\_emp, AVG(salary) AS avg\_salary

FROM emp

WHERE deptno = 20;

//  S**ELECT COUNT(\*) AS no\_of\_emp**: The COUNT(\*) function counts the number of rows (employees) in the emp table, and it is given the alias no\_of\_emp.

 **AVG(salary) AS avg\_salary**: The AVG() function calculates the **average salary** of employees in department number 20, and it is aliased as avg\_salary.

 **FROM emp**: The data is being retrieved from the emp table.

 **WHERE deptno = 20**: This condition filters employees who are in department number 20.

6. List the average salary and minimum salary of employees hiredate-wise for department number "10".

SELECT hiredate, AVG(salary) AS avg\_salary, MIN(salary) AS min\_salary

FROM emp

WHERE deptno = 10

GROUP BY hiredate;

//  **SELECT hiredate**: This selects the hiredate column, which will group the results by hire date.

 **AVG(salary) AS avg\_salary**: This calculates the **average salary** for each hire date and gives it the alias avg\_salary.

 **MIN(salary) AS min\_salary**: This calculates the **minimum salary** for each hire date and gives it the alias min\_salary.

 **FROM emp**: The data is being retrieved from the emp table.

 **WHERE deptno = 10**: This condition filters employees who are in department number 10.

 **GROUP BY hiredate**: This groups the employees by their **hire date**, so the calculations for average and minimum salary are done for each hire date.

7. List employee names and their department names.

SELECT emp.ename, dept.deptname

FROM emp

JOIN dept ON emp.deptno = dept.deptno;

// **JOIN dept ON emp.deptno = dept.deptno**: This performs an **INNER JOIN** between the emp and dept tables on the deptno column. This ensures that each employee is matched with their respective department based on the department number.

8. List the total salary paid to each department.

SELECT dept.deptname, SUM(emp.salary) AS total\_salary

FROM emp

JOIN dept ON emp.deptno = dept.deptno

GROUP BY dept.deptname;

// **SUM(emp.salary) AS total\_salary**: This calculates the **total salary** paid to each department by summing up the salary of all employees in that department.

//  **JOIN dept ON emp.deptno = dept.deptno**: This performs an **INNER JOIN** between the emp and dept tables on the deptno column.

 **GROUP BY dept.deptname**: This groups the results by **department name**, so the total salary is calculated for each department.

9. List details of employees working in the "Development" department.

SELECT emp.\*

FROM emp

JOIN dept ON emp.deptno = dept.deptno

WHERE dept.deptname = 'Development';

//  **OIN dept ON emp.deptno = dept.deptno**: This performs an **INNER JOIN** between the emp and dept tables on the deptno column.

 **WHERE dept.deptname = 'Development'**: This condition filters the results to only include employees who work in the **Development** department.

10. Update the salary of all employees in department number 10 by 5%.

UPDATE emp

SET salary = salary \* 1.05

WHERE deptno = 10;

//  **UPDATE emp**: This updates the emp table.  **SET salary = salary \* 1.05**: This updates the salary column for each employee. The new salary is calculated by multiplying the existing salary by **1.05**, which is a 5% increase.

 **WHERE deptno = 10**: This condition ensures that the salary update only affects employees in department number 10.

PRACTICAL 4TH

--- Create the Employee Table

• Eno as the primary key with a sequence starting at 101.

• Ename as a NOT NULL field.

• Address with a default value of "Nashik."

• Salary with a check constraint ensuring it is greater than 5000.

CREATE TABLE Employee (

Eno INT PRIMARY KEY AUTO\_INCREMENT, Ename VARCHAR(50) NOT NULL,

Address VARCHAR(100) DEFAULT 'Nashik', Join\_Date DATE, Post VARCHAR(50), Salary DECIMAL(10, 2) CHECK (Salary > 5000) );

// The AUTO\_INCREMENT keyword ensures that each new entry automatically gets a unique Eno starting from 101.

-- Set the initial value of Eno to start from 101

ALTER TABLE Employee AUTO\_INCREMENT = 101;

// This ensures that the AUTO\_INCREMENT value for Eno starts at 101, so the first employee inserted will have an Eno of 101.

2. Create the Emp\_Project Table

• Eno as a foreign key referencing the Employee table.

• Project\_name and LOC (Line of Code).

CREATE TABLE Emp\_Project ( Eno INT, Project\_name VARCHAR(100),

LOC INT, FOREIGN KEY (Eno) REFERENCES Employee(Eno) );

// **FOREIGN KEY (Eno) REFERENCES Employee(Eno)**: This ensures that only valid employee numbers from the Employee table can be referenced in this table. This is the foreign key constraint.

INSERT INTO Employee (Ename, Address, Join\_Date, Post, Salary)

VALUES

('John Doe', 'Nashik', '2020-01-15', 'Manager', 7000.00),

('Jane Smith', 'Pune', '2019-03-20', 'Salesman', 6500.00),

('Alex Brown', 'Mumbai', '2021-07-30', 'Developer', 8000.00),

('Emily White', 'Nashik', '2018-12-10', 'Tester', 4000.00),

('Chris Green', 'Delhi', '2022-05-10', 'Salesman', 5500.00);

INSERT INTO Emp\_Project (Eno, Project\_name, LOC)

VALUES

(101, 'Project A', 1200),

(102, 'Project B', 1500),

(103, 'Project C', 2000),

(104, 'Project D', 800),

(105, 'Project E', 950);

3. Create Index on Ename Field of Employee Table

CREATE INDEX idx\_ename ON Employee(Ename);

// This creates an **index** on the Ename column of the Employee table. Indexes help to speed up query performance, especially for queries that filter or sort by the Ename column.

4. Create a View on the Employee Table

• This view will display only Ename, Post, and Salary.

CREATE VIEW Employee\_View AS

SELECT Ename, Post, Salary

FROM Employee;

Queries

1. Find the employee names whose salary is greater than or equal to 6000

SELECT Ename

FROM Employee

WHERE Salary >= 6000;

2. Find the employee names whose salary is 3000

SELECT Ename

FROM Employee

WHERE Salary = 3000;

3. Find name and Join\_Date from the table where salary is less than 5000

SELECT Ename, Join\_Date

FROM Employee

WHERE Salary < 5000;

Practical 5th

//Consider the following two tables:

a) Persons table is as follow:

Pid Last name First name Address City

1 Amit Tiwari kharadi Pune

2 Suresh Mishra Gangapur Nashik

3 Shalu Choudhary savedi Ahmednagar

a) Order Table is as follows:

O\_id Order\_Date Order\_Price P\_id

1 2023-02-11 1000 3

2 2023-03-12 1600 3

3 2023-04-10 700 1

4 2022-07-24 300 1

5 2022-08-30 2000 NULL

a. Apply inner Join on above tables and show the result

b. Apply Left Join (or left outer join) on above tables and show the result.

c. Apply Right Join (or right outer join) on above tables and show the result.//

---------------------------------------------------------------------------------------------

CREATE DATABASE StoreDB;

USE StoreDB; -- Switch to the newly created database

-- Create Persons table

CREATE TABLE Persons (

Pid INT PRIMARY KEY,

LastName VARCHAR(50),

FirstName VARCHAR(50),

Address VARCHAR(100),

City VARCHAR(50)

);

-- Create Orders table

CREATE TABLE Orders (

O\_id INT PRIMARY KEY,

Order\_Date DATE,

Order\_Price DECIMAL(10, 2),

P\_id INT);

-- Insert data into Persons table

INSERT INTO Persons (Pid, LastName, FirstName, Address, City) VALUES

(1, 'Tiwari', 'Amit', 'kharadi', 'Pune'),

(2, 'Mishra', 'Suresh', 'gangapur', 'Nashik'),

(3, 'Choudhary', 'Shalu', 'savedi', 'Ahmednagar');

-- Insert data into Orders table

INSERT INTO Orders (O\_id, Order\_Date, Order\_Price, P\_id) VALUES

(1, '2023-02-11', 1000, 3),

(2, '2023-03-12', 1600, 3),

(3, '2023-04-10', 700, 1),

(4, '2022-07-24', 300, 1),

(5, '2022-08-30', 2000, NULL);

--- Apply INNER JOIN

SELECT

p.Pid,

p.LastName,

p.FirstName,

p.Address,

p.City,

o.O\_id,

o.Order\_Date,

o.Order\_Price

FROM

Persons p

INNER JOIN

Orders o ON p.Pid = o.P\_id;

//The Pid column from the Persons table is matched with the P\_id column from the Orders table. This indicates that the query will pair each person with their corresponding order based on the person’s Pid (person ID) and the P\_id (person ID in the Orders table).

----Apply LEFT JOIN

SELECT

p.Pid,

p.LastName,

p.FirstName,

p.Address,

p.City,

o.O\_id,

o.Order\_Date,

o.Order\_Price

FROM

Persons p

LEFT JOIN

Orders o ON p.Pid = o.P\_id;

//LEFT JOIN Orders o: The LEFT JOIN (or left outer join) combines rows from the Persons table with rows from the Orders table based on the join condition p.Pid = o.P\_id.

The key difference between LEFT JOIN and INNER JOIN is that a LEFT JOIN returns all rows from the left table (in this case, Persons), and the matching rows from the right table (in this case, Orders). If there is no match in the right table, the result will still include the row from the left table, but the columns from the right table (Orders) will have NULL values.

//ON p.Pid = o.P\_id: This specifies the condition for the join. It matches the Pid from the Persons table with the P\_id from the Orders table.

--- Apply RIGHT JOIN

SELECT

p.Pid,

p.LastName,

p.FirstName,

p.Address,

p.City,

o.O\_id,

o.Order\_Date,

o.Order\_Price

FROM

Persons p

RIGHT JOIN

Orders o ON p.Pid = o.P\_id;

//RIGHT JOIN Orders o: The RIGHT JOIN returns all records from the right table (Orders), along with matching records from the left table (Persons). If there is no match for a particular order in the Persons table, the result will still include the order details, but the Persons table columns will be filled with NULL values.

The alias o is used to refer to the Orders table.

//ON p.Pid = o.P\_id: This specifies the condition that links the two tables. The query will match Pid from the Persons table with P\_id from the Orders table

--------------------------------------------------------------------------

INNER JOIN: Displays only those persons who have placed orders.

LEFT JOIN: Displays all persons, showing NULL for orders where there is no match.

RIGHT JOIN: Displays all orders, including those with NULL for persons when there's no match.