1.

1. Table creation:

CREATE TABLE Employee (

employee\_name VARCHAR(50),

street VARCHAR(100),

city VARCHAR(50),

Id INT,

Mobile\_no VARCHAR(20),

PRIMARY KEY (employee\_name)

);

CREATE TABLE Works (

employee\_name VARCHAR(50),

company\_name VARCHAR(50),

salary DECIMAL(10, 2),

PRIMARY KEY (employee\_name, company\_name),

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name)

);

CREATE TABLE Company (

company\_name VARCHAR(50),

city VARCHAR(50),

PRIMARY KEY (company\_name)

);

CREATE TABLE Manages (

employee\_name VARCHAR(50),

manager\_name VARCHAR(50),

PRIMARY KEY (employee\_name),

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),

FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name)

);

1. View creation:

CREATE VIEW EmployeeView AS

SELECT e.employee\_name, e.street, e.city, e.Id, e.Mobile\_no, w.company\_name, w.salary, c.city AS company\_city, m.manager\_name

FROM Employee e

LEFT JOIN Works w ON e.employee\_name = w.employee\_name

LEFT JOIN Company c ON w.company\_name = c.company\_name

LEFT JOIN Manages m ON e.employee\_name = m.employee\_name;

1. Index creation:

CREATE INDEX idx\_employee\_name ON Employee (employee\_name);

CREATE INDEX idx\_company\_name ON Company (company\_name);

1. Sequence creation:

CREATE SEQUENCE employee\_id\_seq START WITH 1 INCREMENT BY 1;

1. Synonym creation:

CREATE SYNONYM emp FOR Employee;

1. Table or view deletion:

DROP TABLE Employee;

DROP VIEW EmployeeView;

1. Committing changes:

COMMIT;

2.

1. Table creation:

CREATE TABLE Employee (

employee\_name VARCHAR(50),

street VARCHAR(100),

city VARCHAR(50),

Id INT,

Mobile\_no VARCHAR(20),

PRIMARY KEY (employee\_name)

);

CREATE TABLE Works (

employee\_name VARCHAR(50),

company\_name VARCHAR(50),

salary DECIMAL(10, 2),

PRIMARY KEY (employee\_name, company\_name),

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name)

);

CREATE TABLE Company (

company\_name VARCHAR(50),

city VARCHAR(50),

PRIMARY KEY (company\_name)

);

CREATE TABLE Manages (

employee\_name VARCHAR(50),

manager\_name VARCHAR(50),

PRIMARY KEY (employee\_name),

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name),

FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name)

);

1. View creation:

CREATE VIEW EmployeeView AS

SELECT e.employee\_name, e.street, e.city, e.Id, e.Mobile\_no, w.company\_name, w.salary, c.city AS company\_city, m.manager\_name

FROM Employee e

LEFT JOIN Works w ON e.employee\_name = w.employee\_name

LEFT JOIN Company c ON w.company\_name = c.company\_name

LEFT JOIN Manages m ON e.employee\_name = m.employee\_name;

1. Index creation:

CREATE INDEX idx\_employee\_name ON Employee (employee\_name);

CREATE INDEX idx\_company\_name ON Company (company\_name);

1. Sequence creation:

CREATE SEQUENCE employee\_id\_seq START WITH 1 INCREMENT BY 1;

1. Synonym creation:

CREATE SYNONYM emp FOR Employee;

1. Table or view deletion:

DROP TABLE Employee;

DROP TABLE Works;

DROP TABLE Company;

DROP TABLE Manages;

1. Committing changes:

COMMIT;

3.

1. Join:

-- Inner Join

SELECT Employee.employee\_name, Works.company\_name, Works.salary

FROM Employee

INNER JOIN Works ON Employee.employee\_name = Works.employee\_name;

-- Left Join

SELECT Employee.employee\_name, Works.company\_name, Works.salary

FROM Employee

LEFT JOIN Works ON Employee.employee\_name = Works.employee\_name;

-- Right Join

SELECT Employee.employee\_name, Works.company\_name, Works.salary

FROM Employee

RIGHT JOIN Works ON Employee.employee\_name = Works.employee\_name;

-- Full Outer Join (not supported in all databases)

SELECT Employee.employee\_name, Works.company\_name, Works.salary

FROM Employee

FULL OUTER JOIN Works ON Employee.employee\_name = Works.employee\_name;

1. Sub-Query:

SELECT employee\_name, company\_name, salary

FROM Works

WHERE employee\_name IN (SELECT employee\_name FROM Manages WHERE manager\_name = 'John');

1. View:

CREATE VIEW EmployeeView AS

SELECT e.employee\_name, e.street, e.city, e.Id, e.Mobile\_no, w.company\_name, w.salary, c.city AS company\_city, m.manager\_name

FROM Employee e

LEFT JOIN Works w ON e.employee\_name = w.employee\_name

LEFT JOIN Company c ON w.company\_name = c.company\_name

LEFT JOIN Manages m ON e.employee\_name = m.employee\_name;

SELECT \* FROM EmployeeView;

4.

-- Create the Student table

CREATE TABLE Student (

Roll\_no INT,

Name VARCHAR(50),

Marks INT,

Remark VARCHAR(10)

);

-- Insert sample data into the Student table

INSERT INTO Student (Roll\_no, Name, Marks, Remark)

VALUES (1, 'John', 80, '');

INSERT INTO Student (Roll\_no, Name, Marks, Remark)

VALUES (2, 'Jane', 30, '');

-- Create a stored procedure to update the remark of a student based on marks

CREATE OR REPLACE PROCEDURE UpdateRemark(p\_Roll\_no IN INT) IS

v\_Marks INT;

BEGIN

-- Get the marks for the given Roll\_no

SELECT Marks INTO v\_Marks

FROM Student

WHERE Roll\_no = p\_Roll\_no;

-- Update the remark based on the marks

IF v\_Marks >= 40 THEN

UPDATE Student

SET Remark = 'pass'

WHERE Roll\_no = p\_Roll\_no;

ELSE

UPDATE Student

SET Remark = 'fail'

WHERE Roll\_no = p\_Roll\_no;

END IF;

-- Commit the changes

COMMIT;

-- Display the updated student record

SELECT \* FROM Student WHERE Roll\_no = p\_Roll\_no;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Student not found.');

END;

/

-- Call the stored procedure to update the remark of student with Roll\_no = 2

BEGIN

UpdateRemark(2);

END;

/

5.

-- Create the Student table

CREATE TABLE Student (

Roll\_no INT,

Name VARCHAR(50),

Address VARCHAR(100),

Marks INT,

Grade VARCHAR(2),

Active CHAR(1)

);

-- Insert sample data into the Student table

INSERT INTO Student (Roll\_no, Name, Address, Marks, Grade, Active)

VALUES (1, 'John', '123 Main Street', 85, 'A', 'Y');

INSERT INTO Student (Roll\_no, Name, Address, Marks, Grade, Active)

VALUES (2, 'Jane', '456 Elm Street', 65, 'B', 'Y');

-- Create the BEFORE DELETE trigger to prevent deletion of active students

CREATE OR REPLACE TRIGGER PreventDeleteActiveStudent

BEFORE DELETE ON Student

FOR EACH ROW

BEGIN

IF :old.Active = 'Y' THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete an active student.');

END IF;

END;

/

-- Create the AFTER DELETE trigger to update the grade of the remaining students

CREATE OR REPLACE TRIGGER UpdateRemainingStudentsGrade

AFTER DELETE ON Student

FOR EACH ROW

BEGIN

UPDATE Student

SET Grade = 'C'

WHERE Roll\_no IN (

SELECT Roll\_no

FROM Student

);

-- Commit the changes

COMMIT;

END;

/

-- Try deleting a non-active student record

DELETE FROM Student WHERE Roll\_no = 1;

-- Try deleting an active student record (will result in an error)

DELETE FROM Student WHERE Roll\_no = 2;

6.

1. Create (Insert) operation:

// Insert a document into a collection

db.students.insertOne({

name: "John Doe",

age: 25,

grade: "A"

});

1. Read (Select) operation:

// Find all documents in a collection

db.students.find();

// Find documents that match specific criteria

db.students.find({ age: { $gt: 20 } }); // Find students older than 20

// Find a single document

db.students.findOne({ name: "John Doe" });

1. Update operation:

// Update a document

db.students.updateOne(

{ name: "John Doe" },

{ $set: { grade: "B" } }

);

// Update multiple documents

db.students.updateMany(

{ age: { $gt: 20 } },

{ $inc: { age: 1 } }

);

1. Delete operation:

// Delete a document

db.students.deleteOne({ name: "John Doe" });

// Delete multiple documents

db.students.deleteMany({ age: { $lt: 20 } });

7.

1. Get complete details from Duty\_allocation:

SELECT \*

FROM Duty\_allocation;

1. Get duty allocation details for Emp\_no 123461 for the month of April 1986:

SELECT \*

FROM Duty\_allocation

WHERE emp\_no = 123461

AND EXTRACT(MONTH FROM day) = 4

AND EXTRACT(YEAR FROM day) = 1986;

1. Find the shift details for employee 'XYZ':

SELECT da.emp\_no, da.day, da.shift

FROM Duty\_allocation da

JOIN Employee e ON da.emp\_no = e.emp\_no

WHERE e.name = 'XYZ';

8.

1. Get employee number of employees working on a project:

SELECT COUNT(\*) AS EmployeeCount

FROM Assigned\_To

WHERE Project\_id = 'project\_id';

1. Get details of employees working on project C353:

SELECT e.Emp\_id, e.Emp\_name

FROM Employee e

JOIN Assigned\_To at ON e.Emp\_id = at.emp\_id

WHERE at.Project\_id = 'C353';

1. Obtain details of employees working on both C353 & C354:

SELECT e.Emp\_id, e.Emp\_name

FROM Employee e

JOIN Assigned\_To at ON e.Emp\_id = at.emp\_id

WHERE at.Project\_id IN ('C353', 'C354')

GROUP BY e.Emp\_id, e.Emp\_name

HAVING COUNT(DISTINCT at.Project\_id) = 2;

9.

1. Find the name of all employees who work for FBC:

SELECT e.employee\_name

FROM Employee e

JOIN Works w ON e.employee\_name = w.employee\_name

JOIN Company c ON w.company\_name = c.company\_name

WHERE c.company\_name = 'FBC';

1. Find the name and cities of all employees who work for FBC:

SELECT e.employee\_name, e.city

FROM Employee e

JOIN Works w ON e.employee\_name = w.employee\_name

JOIN Company c ON w.company\_name = c.company\_name

WHERE c.company\_name = 'FBC';

1. Find the names, street address, and cities of residence of all employees who work for FBC and earn more than $10,000:

SELECT e.employee\_name, e.street, e.city

FROM Employee e

JOIN Works w ON e.employee\_name = w.employee\_name

JOIN Company c ON w.company\_name = c.company\_name

WHERE c.company\_name = 'FBC'

AND w.salary > 10000;