

2. Employee Table

a) Add a salary column.

sql

 Copy code

```
ALTER TABLE employee ADD salary NUMBER;
```

b) Copy the employee table.

sql

 Copy code

```
CREATE TABLE Emp AS SELECT * FROM employee;
```

c) Delete the 2nd row.

sql

 Copy code

```
DELETE FROM employee WHERE ROWNUM = 2;
```

d) Drop the employee table.

sql

 Copy code

```
DROP TABLE employee;
```

e) Trigger to update salary.

sql


 Copy code

```
CREATE OR REPLACE TRIGGER Update_Salary
AFTER INSERT ON employee
FOR EACH ROW
BEGIN
    UPDATE employee SET salary = :NEW.salary * 1.1 WHERE S_No = :NEW.S_No;
END;
```

6. Bank Table

a) Display customers from branch XXXXX.

sql

 Copy code

```
SELECT Cust_Name, Account_No  
FROM Bank  
WHERE Branch = 'XXXXX';
```

b) Display names and account types with balance > 10,000.


sql

 Copy code

```
SELECT Cust_Name, Account_Type  
FROM Bank  
WHERE Balance > 10000;
```

c) Add CONTACTNO column.

sql

 Copy code

```
ALTER TABLE Bank ADD (CONTACTNO VARCHAR2(15));
```

d) Display account details with balance < 1,000.


sql

 Copy code

```
SELECT Account_No, Cust_Name, Branch  
FROM Bank  
WHERE Balance < 1000;
```

e) Trigger for balance < 1,000.

sql

 Copy code

```
CREATE OR REPLACE TRIGGER Check_Balance  
BEFORE UPDATE ON Bank  
FOR EACH ROW  
BEGIN  
    IF :NEW.Balance < 1000 THEN  
        DBMS_OUTPUT.PUT_LINE('Balance below 1000 for Account No ' || :NEW.Account_No);  
    END IF;  
END;
```

5. Event and Prizes

a) Primary and foreign keys.


sql

 Copy code

```
ALTER TABLE Prizes ADD CONSTRAINT PK_Prizes PRIMARY KEY (prizeid);  
ALTER TABLE Minners ADD CONSTRAINT FK_Winners_Prizes FOREIGN KEY (prizeid) REFERENCES Prizes (prizeid);  
ALTER TABLE Minners ADD CONSTRAINT FK_Winners_Participant FOREIGN KEY (playerid) REFERENCES Participant (playerid);
```

b) Player ID with at least one digit.


sql

 Copy code

```
ALTER TABLE Participant ADD CONSTRAINT CHK_PlayerID CHECK (REGEXP_LIKE(playerid, '.*\d+.*'));
```

c) Events with only female prize winners.


sql

 Copy code

```
SELECT DISTINCT e.name  
FROM Event e  
WHERE NOT EXISTS (  
    SELECT 1 FROM Participant p  
    JOIN Minners w ON p.playerid = w.playerid  
    WHERE w.eventid = e.eventid AND p.gender != 'F'  
);
```

d) View of movies with actors and directors.


sql

 Copy code

```
CREATE VIEW Movies_Actors_Directors AS  
SELECT m.Title, a.Name AS Actor, d.Name AS Director  
FROM MOVIES m  
JOIN ACTORS a ON m.Actor_id = a.Actor_id  
JOIN DIRECTORS d ON m.Director_id = d.Director_id;
```

4. Departments and Employees

a) Grant privileges.

 Copy code

```
sql  
  
GRANT SELECT, INSERT ON Employee TO Department;
```

b) Revoke all privileges.

 Copy code


```
sql  
  
REVOKE ALL ON Employee FROM Department;
```

c) Revoke some privileges.

 Copy code

```
sql  
  
REVOKE SELECT, INSERT ON Employee FROM Department;
```

d) Save point.

 Copy code

```
sql  
  
SAVEPOINT my_savepoint;
```

e) Procedure example.


 Copy code

```
sql  
  
CREATE OR REPLACE PROCEDURE Show_Employee_Details AS  
BEGIN  
    FOR rec IN (SELECT * FROM Employee) LOOP  
        DBMS_OUTPUT.PUT_LINE('Name: ' || rec.Name);  
    END LOOP;  
END;
```

9. Account Table

a) Display customers and account numbers from branch XXXXX.

sql

 Copy code

```
SELECT Cust_Name, Account_No
FROM Account
WHERE Branch_Name = 'XXXXX';
```

b) Display names and account types with balance > 10,000.


sql

 Copy code

```
SELECT Cust_Name, Account_Type
FROM Account
WHERE Balance > 10000;
```

c) Add Cust_Date_of_Birth column.

sql

 Copy code

```
ALTER TABLE Account ADD (Cust_Date_of_Birth DATE);
```

d) Display account details with balance < 1,000.

sql

 Copy code

```
SELECT Account_No, Cust_Name, Branch_Name
FROM Account
WHERE Balance < 1000;
```

e) Procedure to show account details.

sql

 Copy code

```
CREATE OR REPLACE PROCEDURE Show_Account_Details AS
BEGIN
    FOR rec IN (SELECT * FROM Account) LOOP
        DBMS_OUTPUT.PUT_LINE('Account No: ' || rec.Account_No || ', Name: ' || rec.Cust_Na
    END LOOP;
END;
```

8. College Database

a) Number of male and female students per semester.

sql

Copy code

```
SELECT Semester, Gender, COUNT(*)  
FROM STUDENT  
GROUP BY Semester, Gender;
```

b) Update Finalmark (average of best two test marks).

sql

Copy code

```
UPDATE MARKS  
SET Finalmark = (GREATEST(Test1, Test2, Test3) + LEAST(GREATEST(Test1, Test2), GREATEST(Te
```

c) Categorize students based on Finalmark.

sql

Copy code

```
UPDATE MARKS  
SET Category = CASE  
    WHEN Finalmark >= 81 THEN 'Outstanding'  
    WHEN Finalmark >= 51 THEN 'Average'  
    ELSE 'Weak'  
END;
```

d) View of Test3 marks for a particular student.

sql

Copy code

```
CREATE VIEW Test3_Marks_View AS  
SELECT RegNo, Subject, Test3  
FROM MARKS  
WHERE RegNo = 'specific_regno'; -- Replace with actual student ID
```

e) Procedure to show student details.

sql

Copy code

```
CREATE OR REPLACE PROCEDURE Show_Student_Details AS  
BEGIN  
    FOR rec IN (SELECT * FROM STUDENT) LOOP  
        DBMS_OUTPUT.PUT_LINE('RegNo: ' || rec.RegNo || ', Name: ' || rec.Name);  
    END LOOP;  
END;
```

Movie Database
List movies directed by 'XXXX'.

Copy code

sql

```
SELECT Title
FROM MOVIES
WHERE Director = 'XXXX';
```

Movies with actors in two or more films.

Copy code

sql

```
SELECT Title
FROM MOVIES
WHERE Actor_id IN (
    SELECT Actor_id
    FROM MOVIES
    GROUP BY Actor_id
    HAVING COUNT(*) > 1
);
```

Actors in movies before 2010 and after 2015.

Copy code

sql

```
SELECT DISTINCT a.Name
FROM ACTORS a
JOIN MOVIES m1 ON a.Actor_id = m1.Actor_id
JOIN MOVIES m2 ON a.Actor_id = m2.Actor_id
WHERE m1.Year < 2010 AND m2.Year > 2015;
```

Create a view of movies with actor and director.

Copy code

sql

```
CREATE VIEW Movies_Actor_Director AS
SELECT m.Title, a.Name AS Actor, d.Name AS Director
FROM MOVIES m
JOIN ACTORS a ON m.Actor_id = a.Actor_id
JOIN DIRECTORS d ON m.Director_id = d.Director_id;
```

Function to get movie title.

Copy code

sql

```
CREATE OR REPLACE FUNCTION Get_Movie_Title (mov_id IN NUMBER) RETURN VARCHAR2 IS
    title VARCHAR2(100);
BEGIN
    SELECT Title INTO title
    FROM MOVIES
    WHERE Movie_id = mov_id;
    RETURN title;
END;
```

```
CREATE OR REPLACE TRIGGER Insert_Employee
AFTER INSERT ON Employee
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('New Employee Added: ' || :NEW.Name);
END;
```

```
CREATE OR REPLACE TRIGGER Delete_Employee
AFTER DELETE ON Employee
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Employee Deleted: ' || :OLD.Name);
END;
```

```
CREATE OR REPLACE TRIGGER Update_Employee
AFTER UPDATE ON Employee
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Employee Updated: ' || :NEW.Name);
END;
```


12. Salesman, Customer, Orders

a) Salesmen with more than one customer.

```
sql Copy code

SELECT s.Name
FROM SALESMAN s
JOIN CUSTOMER c ON s.Salesman_id = c.Salesman_id
GROUP BY s.Name
HAVING COUNT(c.Customer_id) > 1;
```

b) Salesmen with and without customers in their cities.

```
sql Copy code

SELECT s.Name, CASE WHEN c.Customer_id IS NOT NULL THEN 'Has Customers' ELSE 'No Customers'
FROM SALESMAN s
LEFT JOIN CUSTOMER c ON s.Salesman_id = c.Salesman_id AND s.City = c.City;
```

c) View to find the salesman with the highest order of the day.

```
sql Copy code

CREATE VIEW Top_Salesman AS
SELECT s.Name
FROM SALESMAN s
JOIN CUSTOMER c ON s.Salesman_id = c.Salesman_id
JOIN ORDERS o ON c.Customer_id = o.Customer_id
WHERE o.Order_Amount = (SELECT MAX(Order_Amount) FROM ORDERS WHERE Order_Date = SYSDATE);
```

d) Delete salesman with ID 1000 and all his orders.

```
sql Copy code

DELETE FROM ORDERS WHERE Salesman_id = 1000;
DELETE FROM SALESMAN WHERE Salesman_id = 1000;
```

e) Triggers for insert, delete, and update.

```
sql Copy code


CREATE OR REPLACE TRIGGER Insert_Salesman
AFTER INSERT ON SALESMAN
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('New Salesman Added: ' || :NEW.Name);
END;

CREATE OR REPLACE TRIGGER Delete_Salesman
AFTER DELETE ON SALESMAN
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Salesman Deleted: ' || :OLD.Name);
END;

CREATE OR REPLACE TRIGGER Update_Salesman
AFTER UPDATE ON SALESMAN
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Salesman updated: ' || :NEW.Name);
END;
```


10. Customer and Order

a) List names and addresses of customers who ordered products > 500.

```
sql  Copy code


SELECT DISTINCT c.Name, c.Address
FROM CUSTOMER c
JOIN ORDER o ON c.Customer_id = o.Customer_id
WHERE o.Price > 500;
```

b) List product names ordered costing >= 1000.

```
sql  Copy code

SELECT DISTINCT Product_Name
FROM ORDER
WHERE Price >= 1000;
```

c) List product names ordered by customers from Delhi.

```
sql  Copy code


SELECT DISTINCT o.Product_Name
FROM ORDER o
JOIN CUSTOMER c ON o.Customer_id = c.Customer_id
WHERE c.City = 'Delhi';
```

d) Add Email_id column in CUSTOMER table.

```
sql  Copy code

ALTER TABLE CUSTOMER ADD (Email_id VARCHAR2(100));
```

e) Function to get customer address.

```
sql  Copy code

CREATE OR REPLACE FUNCTION Get_Customer_Address (c_id IN NUMBER) RETURN VARCHAR2 IS
    address VARCHAR2(255);
BEGIN
    SELECT Address INTO address
    FROM CUSTOMER
    WHERE Customer_id = c_id;
    RETURN address;
END;
```

1. Library Database

a) Retrieve book details.

sql

Copy code

```
SELECT Book_id, Title, Publisher_Name  
FROM BOOK;
```

b) Borrowers with more than 3 books borrowed between Jan 2017 and Jun 2017.

sql

Copy code

```
SELECT Borrower_ID  
FROM BORROWINGS  
WHERE Borrow_Date BETWEEN '2017-01-01' AND '2017-06-30'  
GROUP BY Borrower_ID  
HAVING COUNT(Book_ID) > 3;
```

c) Delete a book and update other tables.

sql

Copy code

```
DELETE FROM BOOK WHERE Book_id = 1; -- Replace 1 with the actual book id  
DELETE FROM BOOK_AUTHORS WHERE Book_id = 1;
```

d) Create a view of available books.

sql

Copy code

```
CREATE VIEW Available_Books AS  
SELECT Book_id, Title  
FROM BOOK  
WHERE No_of_copies > 0;
```

e) Procedure to display book details by author.

sql

Copy code

```
CREATE OR REPLACE PROCEDURE Get_Books_By_Author (author_name IN VARCHAR2) IS  
BEGIN  
    FOR rec IN (SELECT Title FROM BOOK WHERE Book_id IN (SELECT Book_id FROM BOOK_AUTHORS  
        LOOP  
            DBMS_OUTPUT.PUT_LINE('Title: ' || rec.Title);  
        END LOOP;  
    END;  
END;
```

3. Employee Table with Additional Queries

a) Display students in semester 'xxx'.

sql

 Copy code

```
SELECT Name, Salary
FROM Employee
WHERE Department = 'xxx';
```

b) Lowest-paid employee in each department.

sql

 Copy code

```
SELECT Department, MIN(Salary) AS Lowest_Paid
FROM Employee
GROUP BY Department;
```

c) List employee names in descending order.


sql

 Copy code

```
SELECT Name FROM Employee ORDER BY Name DESC;
```

d) Rename column.


sql

 Copy code

```
ALTER TABLE Employee RENAME COLUMN Job TO Position;
```

e) Insert row using triggers.

sql

 Copy code

```
CREATE OR REPLACE TRIGGER Insert_Trigger
BEFORE INSERT ON Employee
FOR EACH ROW
BEGIN
    IF :NEW.Salary IS NULL THEN
        :NEW.Salary := 1000; -- Default Salary
    END IF;
END;
```

14. Supplier Table

a) Display supplier numbers and names where names start with 'S'.

sql

Copy code

```
SELECT Sup_No, Sup_Name  
FROM Supplier  
WHERE Sup_Name LIKE 'S%';
```

b) Add CONTACTNO column.

sql

Copy code

```
ALTER TABLE Supplier ADD (CONTACTNO VARCHAR2(15));
```

c) Display supplier numbers, names, and item prices for Chennai.

sql

Copy code

```
SELECT Sup_No, Sup_Name, Item_Price  
FROM Supplier  
WHERE City = 'Chennai';
```

d) Create a view showing only supplier numbers and names.

sql

Copy code

```
CREATE VIEW Supplier_View AS  
SELECT Sup_No, Sup_Name  
FROM Supplier;
```

e) Procedure to show supplier details.

sql

Copy code

```
CREATE OR REPLACE PROCEDURE Show_Supplier_Details AS  
BEGIN  
    FOR rec IN (SELECT * FROM Supplier) LOOP  
        DBMS_OUTPUT.PUT_LINE('Supplier No: ' || rec.Sup_No || ', Name: ' || rec.Sup_Name);  
    END LOOP;  
END;
```