

DBMS IT SET-1,2 Answers

SET-1 :

1. Employee Table with Constraints & PLSQL for Salary Hike

a) Create table and perform operations

```
CREATE TABLE Employee (  
    Emp_ID INT PRIMARY KEY,  
    Emp_Name VARCHAR(50) NOT NULL,  
    Dept VARCHAR(30),  
    Salary NUMBER CHECK (Salary > 0),  
    Email VARCHAR(100) UNIQUE  
);
```

-- Insert sample records

```
INSERT INTO Employee VALUES (1, 'Alice', 'Production', 25000, 'alice@company.com');  
INSERT INTO Employee VALUES (2, 'Bob', 'HR', 30000, 'bob@company.com');
```

-- Update

```
UPDATE Employee SET Salary = 28000 WHERE Emp_ID = 1;
```

-- Delete

```
DELETE FROM Employee WHERE Emp_ID = 2;
```

b) PLSQL: Hike salary for production dept

```
BEGIN  
    UPDATE Employee SET Salary = Salary * 1.10  
    WHERE Dept = 'Production';  
END;
```

2. Student Tables & Triggers

a) Create tables with FK constraints

```
CREATE TABLE Department (  
    Dept_ID INT PRIMARY KEY,  
    Dept_Name VARCHAR(50)  
);
```

```
CREATE TABLE Student (  
    Roll_No INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Dept_ID INT,  
    FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)  
);
```

b) Trigger for INSERT/DELETE

```
CREATE OR REPLACE TRIGGER student_log_trigger
AFTER INSERT OR DELETE ON Student
BEGIN
    DBMS_OUTPUT.PUT_LINE('Insert/Delete operation performed on Student table.');
```

3. Library Queries & Trigger

a) Display books related to DBMS

```
SELECT * FROM BOOK
WHERE Title LIKE '%Database Management Systems%';
```

b) Trigger for updating books after purchase

```
CREATE OR REPLACE TRIGGER update_books_after_purchase
AFTER INSERT ON Book_Purchase
FOR EACH ROW
BEGIN
    UPDATE BOOK
    SET No_of_Copies = No_of_Copies + :NEW.Copies
    WHERE Book_ID = :NEW.Book_ID;
END;
```

4. Banking Tables & Procedure

a) Create tables

```
CREATE TABLE Customer (
    Cust_ID INT PRIMARY KEY,
    Name VARCHAR(50)
);

CREATE TABLE Saving_Account (
    Acc_No INT PRIMARY KEY,
    Cust_ID INT,
    Balance NUMBER,
    FOREIGN KEY (Cust_ID) REFERENCES Customer(Cust_ID)
);

CREATE TABLE Loan_Account (
    Loan_ID INT PRIMARY KEY,
    Cust_ID INT,
    Amount NUMBER,
    FOREIGN KEY (Cust_ID) REFERENCES Customer(Cust_ID)
);
```

b) Procedure to insert 50 records

```
CREATE OR REPLACE PROCEDURE insert_50_customers IS
BEGIN
    FOR i IN 1..50 LOOP
        INSERT INTO Customer VALUES (i, 'Customer_' || i);
```

```
END LOOP;  
END;
```

5. Student Mark Database & UDF

a) Create tables and compute marks

```
CREATE TABLE Student (  
    Roll_No INT PRIMARY KEY,  
    Name VARCHAR(50)  
);  
  
CREATE TABLE Subject_Mark (  
    Roll_No INT,  
    Subject VARCHAR(50),  
    Marks INT,  
    FOREIGN KEY (Roll_No) REFERENCES Student(Roll_No)  
);  
  
SELECT MIN(Marks), MAX(Marks), SUM(Marks), AVG(Marks)  
FROM Subject_Mark;
```

b) User-defined function

```
CREATE OR REPLACE FUNCTION calc_percentage(roll INT) RETURN NUMBER IS  
    total NUMBER;  
    percent NUMBER;  
BEGIN  
    SELECT SUM(Marks) INTO total FROM Subject_Mark WHERE Roll_No = roll;  
    percent := total / 5; -- assuming 5 subjects  
    RETURN percent;  
END;
```

6. Railway DB & Seat Trigger

a) Create and display using subqueries

```
CREATE TABLE Train (  
    Train_ID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Total_Seats INT  
);  
  
CREATE TABLE Reservation (  
    Res_ID INT PRIMARY KEY,  
    Train_ID INT,  
    Seats_Booked INT,  
    FOREIGN KEY (Train_ID) REFERENCES Train(Train_ID)  
);  
  
SELECT * FROM Train  
WHERE Train_ID IN (SELECT Train_ID FROM Reservation);
```

b) Trigger to update available seats

```
CREATE OR REPLACE TRIGGER update_seats_after_booking
AFTER INSERT ON Reservation
FOR EACH ROW
BEGIN
    UPDATE Train
    SET Total_Seats = Total_Seats - :NEW.Seats_Booked
    WHERE Train_ID = :NEW.Train_ID;
END;
```

7. Airline Reservation & Procedure**a) Create DB and use JOIN**

```
CREATE TABLE Passenger (
    P_ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Class VARCHAR(20)
);

CREATE TABLE Reservation (
    Res_ID INT PRIMARY KEY,
    P_ID INT,
    FOREIGN KEY (P_ID) REFERENCES Passenger(P_ID)
);

SELECT p.Name, p.Class
FROM Passenger p
JOIN Reservation r ON p.P_ID = r.P_ID
WHERE p.Class IN ('Business', 'Economy');
```

b) Procedure for fare reduction

```
CREATE OR REPLACE PROCEDURE reduce_fare IS
BEGIN
    UPDATE Reservation
    SET Fare = Fare * 0.9; -- 10% discount
END;
```

8. Toy Manufacturing & Procedure**a) Create DB and use HAVING**

```
CREATE TABLE Product (
    Prod_ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Cost NUMBER
);

SELECT Name, AVG(Cost)
FROM Product
GROUP BY Name
```

```
HAVING AVG(Cost) > 100;
```

b) Procedure for insert/update

```
CREATE OR REPLACE PROCEDURE add_product(p_id INT, p_name VARCHAR, p_cost  
NUMBER) IS  
BEGIN  
    INSERT INTO Product VALUES (p_id, p_name, p_cost);  
END;
```

9. Student & Course with Joins & Procedure**a) Create and use JOINS**

```
CREATE TABLE Course (  
    Course_ID INT PRIMARY KEY,  
    Course_Name VARCHAR(50)  
);  
  
CREATE TABLE Student (  
    Roll INT,  
    Name VARCHAR(50),  
    Course_ID INT,  
    FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID)  
);
```

```
SELECT * FROM Student s  
JOIN Course c ON s.Course_ID = c.Course_ID;
```

b) Procedure to display month name

```
CREATE OR REPLACE PROCEDURE get_month_name(month_no INT) IS  
    month_name VARCHAR(20);  
BEGIN  
    SELECT TO_CHAR(TO_DATE(month_no, 'MM'), 'Month') INTO month_name FROM DUAL;  
    DBMS_OUTPUT.PUT_LINE('Month: ' || month_name);  
END;
```

10. DCL Commands & XML for Online Quiz**a) DCL commands**

```
GRANT CREATE, INSERT, UPDATE ON Employee TO user1;  
REVOKE DELETE ON Employee FROM user1;
```

b) XML for quiz

```
<Quiz>  
    <Student>  
        <Name>John</Name>  
        <Result>Passed</Result>  
    </Student>  
</Quiz>
```

11. Transactions & NoSQL Data Models

a) Complex transaction and TCL commands

```
BEGIN
    SAVEPOINT start_point;

    INSERT INTO Account VALUES (101, 'John', 5000);
    UPDATE Account SET Balance = Balance - 1000 WHERE Acc_No = 101;
    INSERT INTO Transaction VALUES (1, 101, 'Debit', 1000);

    COMMIT;
-- If any issue occurs:
-- ROLLBACK TO start_point;
END;
```

b) NoSQL data model examples

// Document-based (MongoDB-like)

```
{
  "_id": "S001",
  "name": "John",
  "marks": [90, 85, 95]
}
```

// Column-based (Cassandra-like)

```
Student_Marks: {
  "S001": { "Math": 90, "Science": 85 },
  "S002": { "Math": 88, "Science": 92 }
}
```

// Graph-based (Neo4j-like)

```
(:Student {name: 'Alice'})-[:ENROLLED_IN]->(:Course {name: 'DBMS'})
```

12. Hostel Management GUI Application

- Use **Python with Tkinter or PyQt** or **Java with Swing/JavaFX**.
- Features:
 - Add/view student records
 - Allocate rooms
 - Generate bills

Basic idea in Python (Tkinter):

```
import tkinter as tk
from tkinter import messagebox
def submit():
    messagebox.showinfo("Success", "Record submitted")
root = tk.Tk()
tk.Label(root, text="Name").pack()
```

```
tk.Entry(root).pack()  
tk.Button(root, text="Submit", command=submit).pack()  
root.mainloop()
```

13. E-Mart Grocery Shop Application

- Use any tech stack (Python/Flask, Node/Express, Java/Spring).
- Key modules:
 - Product catalog
 - Cart & checkout
 - Inventory management
 - Payment simulation

Example DB schema:

```
CREATE TABLE Product (  
  ID INT PRIMARY KEY,  
  Name VARCHAR(50),  
  Price NUMBER  
);
```

```
CREATE TABLE Orders (  
  Order_ID INT PRIMARY KEY,  
  Product_ID INT,  
  Quantity INT,  
  FOREIGN KEY (Product_ID) REFERENCES Product(ID)  
);
```

14. Small Finance Corporation Software

- Features:
 - Customer registration
 - Loan processing
 - EMI calculation
 - Transaction records

Sample EMI logic (Python):

```
def calculate_emi(p, r, t):  
    r = r / (12 * 100)  
    emi = p * r * ((1 + r)**t) / (((1 + r)**t) - 1)  
    return emi
```

15. E-Services for Revenue Department

- Services:
 - Property tax filing
 - Income certificate requests
 - Online grievance redressal

Example entities:

```
CREATE TABLE Service_Request (  
  Req_ID INT PRIMARY KEY,  
  Citizen_Name VARCHAR(100),  
  Service_Type VARCHAR(50),  
  Status VARCHAR(20)  
);
```

16. Society Financial Management Software

- Modules:
 - Maintenance billing
 - Member contributions
 - Expense tracking
 - Audit reports

Basic table:

```
CREATE TABLE Expense (  
  ID INT PRIMARY KEY,  
  Description VARCHAR(100),  
  Amount NUMBER,  
  Date DATE  
);
```

17. Property Management in eMall

- Entities:
 - Store rentals
 - Property owners
 - Lease agreements

DB schema:

```
CREATE TABLE Shop (  
  Shop_ID INT PRIMARY KEY,  
  Owner_Name VARCHAR(50),  
  Rent NUMBER  
);
```


18. Tourism Management System

- Features:
 - Tour packages
 - Booking system
 - Customer feedback
 - Payment integration

Tour package example:

```
CREATE TABLE Tour_Package (  
  ID INT PRIMARY KEY,  
  Destination VARCHAR(100),  
  Cost NUMBER,  
  Duration INT  
);
```

19. Retail Business Application

- Modules:
 - Inventory
 - Billing
 - Customer management
 - Reports

Inventory table:

```
CREATE TABLE Inventory (  
  Product_ID INT PRIMARY KEY,  
  Product_Name VARCHAR(50),  
  Quantity INT,  
  Price NUMBER  
);
```

20. Online Trading System Software

- Features:
 - Buy/sell orders
 - User portfolio
 - Live price tracking
 - Order history

Order table:

```
CREATE TABLE Trade_Order (  
    Order_ID INT PRIMARY KEY,  
    User_ID INT,  
    Stock_Symbol VARCHAR(10),  
    Order_Type VARCHAR(10), -- Buy/Sell  
    Quantity INT,  
    Price NUMBER  
);
```

SET-2 :

1. Library Database

Schema:

```
BOOK(Book_id, Title, Publisher_Name, Pub_Year, No_of_copies)  
BOOK_AUTHORS(Book_id, Author_Name)  
PUBLISHER(Name, Address, Phone)
```

a)

```
SELECT B.Book_id, B.Title, B.Publisher_Name, A.Author_Name  
FROM BOOK B  
JOIN BOOK_AUTHORS A ON B.Book_id = A.Book_id;
```

b) (Assume a table: *BORROWERS*(Borrower_id, Book_id, Date_Borrowed))

```
SELECT Borrower_id  
FROM BORROWERS  
WHERE Date_Borrowed BETWEEN '01-JAN-2017' AND '30-JUN-2017'  
GROUP BY Borrower_id  
HAVING COUNT(Book_id) > 3;
```

c)

```
DELETE FROM BOOK_AUTHORS WHERE Book_id = 101;  
DELETE FROM BOOK WHERE Book_id = 101;
```

d)

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

e)

```
CREATE OR REPLACE PROCEDURE GetBooksByAuthor(p_author IN VARCHAR2) IS  
BEGIN  
    FOR rec IN (  
        SELECT B.* FROM BOOK B  
        JOIN BOOK_AUTHORS A ON B.Book_id = A.Book_id  
        WHERE A.Author_Name = p_author
```

```
) LOOP
  DBMS_OUTPUT.PUT_LINE('Book: ' || rec.Title);
END LOOP;
END;
```

2. Employee Table

```
CREATE TABLE employee (
  S_No NUMBER,
  Name VARCHAR2(50),
  Designation VARCHAR2(50),
  Branch VARCHAR2(50)
);
```

a)

```
ALTER TABLE employee ADD Salary NUMBER;
```

b)

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

c)

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

d)

```
DROP TABLE employee;
```

e)

```
CREATE OR REPLACE TRIGGER emp_salary_update
BEFORE UPDATE ON Emp
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('Salary Updated');
END;
```

3. Employee Table with Salary

```
CREATE TABLE Employee (
  Emp_no NUMBER,
  Emp_name VARCHAR2(50),
  Emp_dept VARCHAR2(50),
  Job VARCHAR2(50),
  Mgr NUMBER,
  Sal NUMBER
);
```

a)

```
SELECT Emp_name, Sal FROM Employee
```

```
WHERE Emp_dept = 'xxx'  
GROUP BY Emp_name, Sal;
```

b)

```
SELECT Emp_dept, MIN(Sal) AS Lowest_Sal  
FROM Employee  
GROUP BY Emp_dept;
```

c)

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

d)

```
ALTER TABLE Employee RENAME COLUMN Emp_name TO Employee_Name;
```

e)

```
CREATE OR REPLACE TRIGGER insert_trigger  
BEFORE INSERT ON Employee  
FOR EACH ROW  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('New employee inserted: ' || :NEW.Emp_name);  
END;
```

4. EMPLOYEES and DEPARTMENTS

```
-- Assumed: EMPLOYEES(emp_id, emp_name, emp_salary, dept_no)  
--           DEPARTMENTS(dept_no, dept_name, dept_location)
```

a)

```
GRANT SELECT, INSERT ON EMPLOYEES TO DEPARTMENTS;
```

b)

```
REVOKE ALL ON EMPLOYEES FROM DEPARTMENTS;
```

c)

```
REVOKE INSERT ON EMPLOYEES FROM DEPARTMENTS;
```

d)

```
SAVEPOINT before_update;
```

e)

```
CREATE OR REPLACE PROCEDURE EmpDetailsByDept(p_dept_no NUMBER) IS  
BEGIN  
    FOR rec IN (  
        SELECT * FROM EMPLOYEES WHERE dept_no = p_dept_no  
    ) LOOP  
        DBMS_OUTPUT.PUT_LINE('Name: ' || rec.emp_name || ', Salary: ' || rec.emp_salary);  
    END LOOP;  
END;
```

```
END LOOP;  
END;
```

5. Event Tables

-- Tables: Event, Participant, Prizes, Winners

a)

-- Primary Keys: eventid, playerid, prizeid

-- Foreign Keys:

```
ALTER TABLE Participant ADD CONSTRAINT fk_event FOREIGN KEY (eventid)  
REFERENCES Event(eventid);
```

```
ALTER TABLE Winners ADD CONSTRAINT fk_prize FOREIGN KEY (prizeid) REFERENCES  
Prizes(prizeid);
```

b)

```
CHECK (REGEXP_LIKE(playerid, '[0-9]'))
```

c)

```
SELECT DISTINCT E.name  
FROM Event E  
JOIN Prizes P ON E.eventid = P.eventid  
JOIN Winners W ON P.prizeid = W.prizeid  
JOIN Participant Pa ON W.playerid = Pa.playerid  
GROUP BY E.name  
HAVING MIN(Pa.gender) = 'F' AND MAX(Pa.gender) = 'F';
```

d)

```
CREATE VIEW FirstPrizeWinners AS  
SELECT Pa.name, E.name AS Event_Name  
FROM Participant Pa  
JOIN Winners W ON Pa.playerid = W.playerid  
JOIN Prizes P ON W.prizeid = P.prizeid  
JOIN Event E ON E.eventid = P.eventid  
WHERE P.rank = 1  
WITH CHECK OPTION;
```

e)

```
CREATE OR REPLACE TRIGGER prize_insert_trigger  
AFTER INSERT ON Event  
FOR EACH ROW  
BEGIN  
    INSERT INTO Prizes VALUES (prize_seq.NEXTVAL, 1500, :NEW.eventid, 1,  
EXTRACT(YEAR FROM SYSDATE));  
    INSERT INTO Prizes VALUES (prize_seq.NEXTVAL, 1000, :NEW.eventid, 2,  
EXTRACT(YEAR FROM SYSDATE));  
    INSERT INTO Prizes VALUES (prize_seq.NEXTVAL, 500, :NEW.eventid, 3, EXTRACT(YEAR  
FROM SYSDATE));  
END;
```

Great! Here's the **second set (Questions 11 to 20)** with **simple SQL/PLSQL programs**:

11. SALESMAN, CUSTOMER, ORDERS

Tables:

SALESMAN(Salesman_id, Name, City, Commission)

CUSTOMER(Customer_id, Cust_Name, City, Grade, Salesman_id)

ORDERS(Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id)

a)

```
SELECT Salesman_id, Name
FROM SALESMAN
WHERE Salesman_id IN (
  SELECT Salesman_id
  FROM CUSTOMER
  GROUP BY Salesman_id
  HAVING COUNT(Customer_id) > 1
);
```

b)

```
SELECT Name, City FROM SALESMAN
UNION
SELECT Name, City FROM SALESMAN
WHERE City NOT IN (
  SELECT DISTINCT City FROM CUSTOMER
);
```

c)

```
CREATE VIEW TopSalesman AS
SELECT S.Name
FROM SALESMAN S
JOIN ORDERS O ON S.Salesman_id = O.Salesman_id
WHERE O.Purchase_Amt = (
  SELECT MAX(Purchase_Amt) FROM ORDERS
);
```

d)

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

e)

```
CREATE OR REPLACE TRIGGER delete_order_check
BEFORE DELETE ON SALESMAN
FOR EACH ROW
BEGIN
  DELETE FROM ORDERS WHERE Salesman_id = :OLD.Salesman_id;
END;
```

12. GUI-based Inventory Management System

(Concept only: GUI via any frontend tool like Python Tkinter or JavaFX, with these DB features)

DB Features:

- Table: Inventory(Item_id, Item_name, Quantity, Price)
- Insert, Update, Delete operations
- Triggers to log low-stock
- Views for summary

Example SQL:

```
CREATE TABLE Inventory (  
  Item_id INT PRIMARY KEY,  
  Item_name VARCHAR2(50),  
  Quantity NUMBER,  
  Price NUMBER  
);
```

13. XML Student Profile

XML Example:

```
<student>  
  <RegNo>101</RegNo>  
  <Name>John</Name>  
  <Address>City A</Address>  
  <Phone>1234567890</Phone>  
</student>
```

XSD (Schema):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="student">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="RegNo" type="xs:int"/>  
        <xs:element name="Name" type="xs:string"/>  
        <xs:element name="Address" type="xs:string"/>  
        <xs:element name="Phone" type="xs:string"/>  
      </xs:sequence>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

14. GUI-based Eseva App

Use:

- Tables for Citizen, Complaint, Officer

- CRUD operations
 - Stored procedure for complaint status
 - Triggers for status change notification
-

15. Employee Triggers

```
CREATE TABLE Employee (  
    EmpID INT PRIMARY KEY,  
    EmpName VARCHAR(50),  
    Salary NUMBER  
);
```

-- Insert Trigger

```
CREATE OR REPLACE TRIGGER emp_insert  
AFTER INSERT ON Employee  
FOR EACH ROW  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('New Employee Added');  
END;
```

-- Update Trigger

```
CREATE OR REPLACE TRIGGER emp_update  
AFTER UPDATE ON Employee  
FOR EACH ROW  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Employee Salary Updated');  
END;
```

-- Delete Trigger

```
CREATE OR REPLACE TRIGGER emp_delete  
AFTER DELETE ON Employee  
FOR EACH ROW  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Employee Deleted');  
END;
```

16. Supplier Table

```
CREATE TABLE Supplier (  
    Sup_No INT,  
    Sup_Name VARCHAR(50),  
    Item_Supplied VARCHAR(50),  
    Item_Price NUMBER,  
    City VARCHAR(50)  
);
```

a)

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


b)

```
ALTER TABLE Supplier ADD ContactNo VARCHAR(15);
```

c)

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

d)

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

e)

```
CREATE OR REPLACE PROCEDURE show_suppliers IS  
BEGIN  
  FOR rec IN (SELECT * FROM Supplier) LOOP  
    DBMS_OUTPUT.PUT_LINE(rec.Sup_Name || ' - ' || rec.City);  
  END LOOP;  
END;
```

17-20: GUI Applications (Conceptual)

17. Banking System

- Account table
- Deposit/Withdraw logic
- Trigger to prevent balance < 0
- GUI via Python/Java

18. Payroll System

- Employees, Salary tables
- View salary slips
- Triggers for raise
- GUI for employee login

19. Movie Ticket System

- Tables: Movies, Shows, Tickets
- View showtimes
- Book tickets
- GUI calendar/date picker

20. SuperMarket Stock

- Product table

- Views for low stock
 - Reorder logic
 - GUI for point of sale
-