1. SQL Aggregate Functions with Examples

SQL CODE

```
-- Step 1: Create Table
CREATE TABLE EMPLOYEES (
   EMP_ID     NUMBER PRIMARY KEY,
   EMP_NAME   VARCHAR2(50),
   DEPARTMENT VARCHAR2(30),
   SALARY     NUMBER(10,2)
);
-- Step 2: Insert Records
INSERT INTO EMPLOYEES VALUES (101, 'Ravi Kumar',    'IT',      65000);
INSERT INTO EMPLOYEES VALUES (102, 'Priya Sharma',  'HR',      52000);
INSERT INTO EMPLOYEES VALUES (103, 'Arjun Nair',    'IT',      75000);
INSERT INTO EMPLOYEES VALUES (104, 'Sneha Rao',     'Finance', 80000);
INSERT INTO EMPLOYEES VALUES (105, 'Vikram Singh',  'HR',      48000);
INSERT INTO EMPLOYEES VALUES (106, 'Deepa Menon',   'Finance', 91000);
INSERT INTO EMPLOYEES VALUES (107, 'Kiran Patel',   'IT',      70000);
COMMIT;
-- Step 3: COUNT – Total number of employees
SELECT COUNT(*) AS TOTAL_EMPLOYEES FROM EMPLOYEES;

-- Step 4: SUM – Total salary
SELECT SUM(SALARY) AS TOTAL_SALARY FROM EMPLOYEES;

-- Step 5: AVG – Average salary
SELECT ROUND(AVG(SALARY), 2) AS AVERAGE_SALARY FROM EMPLOYEES;

-- Step 6: MAX – Highest salary
SELECT MAX(SALARY) AS MAX_SALARY FROM EMPLOYEES;

-- Step 7: MIN – Lowest salary
SELECT MIN(SALARY) AS MIN_SALARY FROM EMPLOYEES;

-- Step 8: GROUP BY – Department-wise statistics
SELECT DEPARTMENT,
      COUNT(*)            AS EMP_COUNT,
      SUM(SALARY)         AS DEPT_TOTAL,
      ROUND(AVG(SALARY), 2) AS DEPT_AVG,
      MAX(SALARY)         AS HIGHEST,
      MIN(SALARY)         AS LOWEST
FROM   EMPLOYEES
GROUP  BY DEPARTMENT
ORDER  BY DEPARTMENT;
```

OUTPUT

```
TOTAL_EMPLOYEES
---------------
      7
TOTAL_SALARY
------------
    481000
AVERAGE_SALARY
--------------
  68714.29
MAX_SALARY
----------
   91000
MIN_SALARY
----------
   48000
DEPARTMENT  EMP_COUNT  DEPT_TOTAL  DEPT_AVG   HIGHEST  LOWEST
----------  ---------  ----------  ---------  -------  ------
Finance         2       171000     85500.00    91000   80000
HR              2       100000     50000.00    52000   48000
IT              3       210000     70000.00    75000   65000
```

2. PL/SQL Program for Electricity Bill Generation using Cursor

PL/SQL CODE

```
-- Step 1: Create Table
CREATE TABLE ELECTRICITY (
   CUST_ID    NUMBER PRIMARY KEY,
   CUST_NAME  VARCHAR2(50),
   UNITS      NUMBER,
   BILL_AMT   NUMBER(10,2)
);

-- Step 2: Insert Records
INSERT INTO ELECTRICITY VALUES (1, 'Ramesh',  85,  NULL);
INSERT INTO ELECTRICITY VALUES (2, 'Suresh',  150, NULL);
INSERT INTO ELECTRICITY VALUES (3, 'Kavitha', 260, NULL);
INSERT INTO ELECTRICITY VALUES (4, 'Anil',    320, NULL);
INSERT INTO ELECTRICITY VALUES (5, 'Meena',   45,  NULL);
COMMIT;

-- Step 3: PL/SQL Block with Cursor
DECLARE
   v_cid    ELECTRICITY.CUST_ID%TYPE;
   v_name   ELECTRICITY.CUST_NAME%TYPE;
   v_units  ELECTRICITY.UNITS%TYPE;
   v_bill   NUMBER(10,2);

   CURSOR c_elec IS
      SELECT CUST_ID, CUST_NAME, UNITS FROM ELECTRICITY;

BEGIN
   OPEN c_elec;
   DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
   DBMS_OUTPUT.PUT_LINE('        ELECTRICITY BILL REPORT          ');
   DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
   DBMS_OUTPUT.PUT_LINE('ID  Name        Units   Bill Amount (Rs.)');
   DBMS_OUTPUT.PUT_LINE('-----------------------------------------');

   LOOP
      FETCH c_elec INTO v_cid, v_name, v_units;
      EXIT WHEN c_elec%NOTFOUND;

      -- Slab-wise bill computation
      IF    v_units <= 100 THEN
         v_bill := v_units * 1.50;
      ELSIF v_units <= 200 THEN
         v_bill := (100 * 1.50) + ((v_units - 100) * 2.50);
      ELSIF v_units <= 300 THEN
         v_bill := (100 * 1.50) + (100 * 2.50) + ((v_units - 200) * 4.00);
      ELSE
         v_bill := (100 * 1.50) + (100 * 2.50) + (100 * 4.00)
                + ((v_units - 300) * 6.00);
      END IF;

      -- Add fixed service charge
      v_bill := v_bill + 50;
```

```
        -- Update bill in table
        UPDATE ELECTRICITY
        SET    BILL_AMT = v_bill
        WHERE  CUST_ID  = v_cid;

        DBMS_OUTPUT.PUT_LINE(
           v_cid || '   ' || RPAD(v_name, 10) ||
           ' ' || LPAD(v_units, 5) ||
           '   Rs. ' || v_bill);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
    CLOSE c_elec;
    COMMIT;
END;
/
```

OUTPUT

```
-------------------------------------------
        ELECTRICITY BILL REPORT
-------------------------------------------
ID  Name       Units   Bill Amount (Rs.)
-------------------------------------------
1   Ramesh       85   Rs. 177.5
2   Suresh      150   Rs. 275
3   Kavitha     260   Rs. 590
4   Anil        320   Rs. 770
5   Meena        45   Rs. 117.5
-------------------------------------------

PL/SQL procedure successfully completed.
```

3. PL/SQL Program to Generate ICET Rank Card Report with Grade Assignment

PL/SQL CODE

```
-- Step 1: Create Table
CREATE TABLE ICET_STUDENTS (
   HALL_TICKET  VARCHAR2(15) PRIMARY KEY,
   STUDENT_NAME VARCHAR2(50),
   MATHS        NUMBER(3),
   REASONING    NUMBER(3),
   COMM         NUMBER(3)
);

-- Step 2: Insert Sample Data
INSERT INTO ICET_STUDENTS VALUES ('ICET2024001','Arjun Reddy',    92, 88, 90);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024002','Priya Lakshmi',  76, 80, 74);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024003','Kiran Babu',     55, 62, 58);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024004','Divya Nair',     42, 38, 45);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024005','Suresh Goud',    30, 28, 35);
COMMIT;

-- Step 3: PL/SQL Block
DECLARE
   v_hall   ICET_STUDENTS.HALL_TICKET%TYPE;
   v_name   ICET_STUDENTS.STUDENT_NAME%TYPE;
   v_math   ICET_STUDENTS.MATHS%TYPE;
   v_res    ICET_STUDENTS.REASONING%TYPE;
   v_comm   ICET_STUDENTS.COMM%TYPE;
   v_total  NUMBER;
   v_grade  VARCHAR2(5);

   CURSOR c_icet IS
      SELECT HALL_TICKET, STUDENT_NAME, MATHS, REASONING, COMM
      FROM   ICET_STUDENTS
      ORDER  BY HALL_TICKET;
BEGIN
   OPEN c_icet;
   DBMS_OUTPUT.PUT_LINE('================================================');
   DBMS_OUTPUT.PUT_LINE('       ICET 2024 RANK CARD REPORT          ');
   DBMS_OUTPUT.PUT_LINE('================================================');
   DBMS_OUTPUT.PUT_LINE('HallTicket     Name              Math Reas Comm Total Grade');
   DBMS_OUTPUT.PUT_LINE('------------------------------------------------------------');

   LOOP
      FETCH c_icet INTO v_hall, v_name, v_math, v_res, v_comm;
      EXIT WHEN c_icet%NOTFOUND;

      v_total := v_math + v_res + v_comm;

      IF    v_total >= 270 THEN  v_grade := 'O';
      ELSIF v_total >= 225 THEN  v_grade := 'A+';
      ELSIF v_total >= 180 THEN  v_grade := 'A';
      ELSIF v_total >= 150 THEN  v_grade := 'B';
      ELSIF v_total >= 120 THEN  v_grade := 'C';
      ELSE                       v_grade := 'F';
      END IF;

      DBMS_OUTPUT.PUT_LINE(
         RPAD(v_hall, 15) || RPAD(v_name, 18) ||
         LPAD(v_math, 4)  || LPAD(v_res, 5)   ||
         LPAD(v_comm, 5)  || LPAD(v_total, 6) ||
         '  ' || v_grade);
   END LOOP;
```

```
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------------');
    CLOSE c_icet;
END;
/
```

OUTPUT

```
================================================
      ICET 2024 RANK CARD REPORT
================================================
HallTicket     Name            Math Reas Comm Total Grade
----------------------------------------------------------------
ICET2024001    Arjun Reddy       92   88   90   270  O
ICET2024002    Priya Lakshmi      76   80   74   230  A+
ICET2024003    Kiran Babu        55   62   58   175  B
ICET2024004    Divya Nair        42   38   45   125  C
ICET2024005    Suresh Goud        30   28   35    93  F
----------------------------------------------------------------

PL/SQL procedure successfully completed.
```

4. Database Trigger to Track Bank Transactions

PL/SQL CODE

```
-- Step 1: Create Bank Accounts Table
CREATE TABLE BANK_ACCOUNTS (
   ACC_NO    VARCHAR2(15) PRIMARY KEY,
   ACC_NAME  VARCHAR2(50),
   BALANCE   NUMBER(12,2)
);

-- Step 2: Create Transaction Log Table
CREATE TABLE TRANSACTION_LOG (
   LOG_ID      NUMBER PRIMARY KEY,
   ACC_NO      VARCHAR2(15),
   TXN_TYPE    VARCHAR2(15),
   AMOUNT      NUMBER(12,2),
   OLD_BAL     NUMBER(12,2),
   NEW_BAL     NUMBER(12,2),
   TXN_TIME    TIMESTAMP DEFAULT SYSTIMESTAMP
);

-- Step 3: Create Sequence
CREATE SEQUENCE TXN_SEQ START WITH 1 INCREMENT BY 1;

-- Step 4: Insert Sample Accounts
INSERT INTO BANK_ACCOUNTS VALUES ('SB100001', 'Ravi Kumar',   50000);
INSERT INTO BANK_ACCOUNTS VALUES ('SB100002', 'Priya Sharma', 30000);
INSERT INTO BANK_ACCOUNTS VALUES ('SB100003', 'Arjun Nair',   75000);
COMMIT;

-- Step 5: Create Trigger
CREATE OR REPLACE TRIGGER trg_bank_txn
   AFTER UPDATE OF BALANCE ON BANK_ACCOUNTS
   FOR EACH ROW
DECLARE
   v_txn_type  VARCHAR2(15);
   v_amount    NUMBER(12,2);
BEGIN
   IF :NEW.BALANCE > :OLD.BALANCE THEN
      v_txn_type := 'DEPOSIT';
      v_amount   := :NEW.BALANCE - :OLD.BALANCE;
   ELSIF :NEW.BALANCE < :OLD.BALANCE THEN
      v_txn_type := 'WITHDRAWAL';
      v_amount   := :OLD.BALANCE - :NEW.BALANCE;
   ELSE
      v_txn_type := 'NO CHANGE';
      v_amount   := 0;
   END IF;

   INSERT INTO TRANSACTION_LOG
      (LOG_ID, ACC_NO, TXN_TYPE, AMOUNT, OLD_BAL, NEW_BAL)
   VALUES
      (TXN_SEQ.NEXTVAL, :NEW.ACC_NO,
       v_txn_type, v_amount, :OLD.BALANCE, :NEW.BALANCE);
END;
/

-- Step 6: Test the Trigger
UPDATE BANK_ACCOUNTS SET BALANCE = 60000 WHERE ACC_NO = 'SB100001'; -- Deposit
UPDATE BANK_ACCOUNTS SET BALANCE = 20000 WHERE ACC_NO = 'SB100002'; -- Withdrawal
UPDATE BANK_ACCOUNTS SET BALANCE = 95000 WHERE ACC_NO = 'SB100003'; -- Deposit
COMMIT;
```

```
-- Step 7: Verify Log
SELECT LOG_ID, ACC_NO, TXN_TYPE, AMOUNT, OLD_BAL, NEW_BAL,
      TO_CHAR(TXN_TIME,'DD-MON-YYYY HH24:MI:SS') AS TXN_TIME
FROM   TRANSACTION_LOG
ORDER  BY LOG_ID;
```

OUTPUT

```
Trigger TRG_BANK_TXN created.

3 rows updated.

LOG_ID  ACC_NO    TXN_TYPE    AMOUNT   OLD_BAL   NEW_BAL  TXN_TIME
------ -------- ---------- ------- -------- ------- --------------------
  1    SB100001  DEPOSIT     10000    50000     60000    01-JAN-2025 10:15:32
  2    SB100002  WITHDRAWAL  10000    30000     20000    01-JAN-2025 10:15:32
  3    SB100003  DEPOSIT     20000    75000     95000    01-JAN-2025 10:15:32
```

5. Exception Handling in PL/SQL with Examples

PL/SQL CODE

```
-- Setup Table
CREATE TABLE DEMO_STUDENTS (
   STU_ID   NUMBER PRIMARY KEY,
   STU_NAME VARCHAR2(40),
   MARKS    NUMBER(3)
);
INSERT INTO DEMO_STUDENTS VALUES (1, 'Ankit',   85);
INSERT INTO DEMO_STUDENTS VALUES (2, 'Bhavana', 72);
INSERT INTO DEMO_STUDENTS VALUES (3, 'Charan',  90);
COMMIT;

-- EXAMPLE 1: NO_DATA_FOUND
DECLARE
   v_name DEMO_STUDENTS.STU_NAME%TYPE;
BEGIN
   SELECT STU_NAME INTO v_name
   FROM   DEMO_STUDENTS
   WHERE  STU_ID = 999;   -- Non-existent ID
   DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('ERROR: No student found with the given ID.');
END;
/

-- EXAMPLE 2: TOO_MANY_ROWS
DECLARE
   v_name DEMO_STUDENTS.STU_NAME%TYPE;
BEGIN
   SELECT STU_NAME INTO v_name
   FROM   DEMO_STUDENTS;  -- Returns 3 rows into scalar
EXCEPTION
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('ERROR: Query returned more than one row.');
END;
/

-- EXAMPLE 3: ZERO_DIVIDE
DECLARE
   v_result NUMBER;
BEGIN
   v_result := 100 / 0;
   DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('ERROR: Division by zero is not allowed.');
END;
/

-- EXAMPLE 4: User-Defined Exception
DECLARE
   e_low_marks  EXCEPTION;
   v_marks      NUMBER := 28;
BEGIN
   IF v_marks < 35 THEN
      RAISE e_low_marks;
   END IF;
   DBMS_OUTPUT.PUT_LINE('Student Passed.');
EXCEPTION
   WHEN e_low_marks THEN
```

```
        DBMS_OUTPUT.PUT_LINE('ALERT: Student has scored below minimum pass marks (35).');
END;
/

-- EXAMPLE 5: WHEN OTHERS with SQLCODE / SQLERRM
DECLARE
    v_num NUMBER := 'ABC';   -- Triggers VALUE_ERROR
BEGIN
    DBMS_OUTPUT.PUT_LINE('Value: ' || v_num);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Unexpected Error!');
        DBMS_OUTPUT.PUT_LINE('Error Code : ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('Error Msg  : ' || SQLERRM);
END;
/
```

OUTPUT

```
-- Example 1:
ERROR: No student found with the given ID.

-- Example 2:
ERROR: Query returned more than one row.

-- Example 3:
ERROR: Division by zero is not allowed.

-- Example 4:
ALERT: Student has scored below minimum pass marks (35).

-- Example 5:
Unexpected Error!
Error Code : -6502
Error Msg  : ORA-06502: PL/SQL: numeric or value error: character to number conversion error
```

6. SQL Commands using Master-Child Tables

SQL CODE

```
-- Step 1: Create Master Table
CREATE TABLE DEPARTMENTS (
   DEPT_ID   NUMBER PRIMARY KEY,
   DEPT_NAME VARCHAR2(40),
   LOCATION  VARCHAR2(40)
);

-- Step 2: Create Child Table with Foreign Key
CREATE TABLE EMP_DETAIL (
   EMP_ID    NUMBER PRIMARY KEY,
   EMP_NAME  VARCHAR2(50),
   SALARY    NUMBER(10,2),
   DEPT_ID   NUMBER,
   CONSTRAINT fk_dept FOREIGN KEY (DEPT_ID)
      REFERENCES DEPARTMENTS(DEPT_ID)
      ON DELETE CASCADE
);

-- Step 3: Insert into Master Table
INSERT INTO DEPARTMENTS VALUES (10, 'Information Technology', 'Hyderabad');
INSERT INTO DEPARTMENTS VALUES (20, 'Human Resources',        'Bangalore');
INSERT INTO DEPARTMENTS VALUES (30, 'Finance',                'Mumbai');
INSERT INTO DEPARTMENTS VALUES (40, 'Research & Dev',         'Pune');
COMMIT;

-- Step 4: Insert into Child Table
INSERT INTO EMP_DETAIL VALUES (101, 'Ravi Kumar',   65000, 10);
INSERT INTO EMP_DETAIL VALUES (102, 'Priya Sharma', 52000, 20);
INSERT INTO EMP_DETAIL VALUES (103, 'Arjun Nair',   75000, 10);
INSERT INTO EMP_DETAIL VALUES (104, 'Sneha Rao',    80000, 30);
INSERT INTO EMP_DETAIL VALUES (105, 'Kiran Patel',  70000, 10);
COMMIT;

-- Step 5: INNER JOIN - Employees with Department
SELECT E.EMP_ID, E.EMP_NAME, E.SALARY,
      D.DEPT_NAME, D.LOCATION
FROM   EMP_DETAIL   E
INNER  JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID
ORDER  BY E.EMP_ID;

-- Step 6: LEFT JOIN - All Departments (even without employees)
SELECT D.DEPT_ID, D.DEPT_NAME,
      NVL(E.EMP_NAME, 'No Employees') AS EMP_NAME
FROM   DEPARTMENTS D
LEFT   JOIN EMP_DETAIL E ON D.DEPT_ID = E.DEPT_ID
ORDER  BY D.DEPT_ID;

-- Step 7: ON DELETE CASCADE Demo
DELETE FROM DEPARTMENTS WHERE DEPT_ID = 10;
COMMIT;

SELECT * FROM EMP_DETAIL;  -- Dept 10 employees auto-deleted
```

OUTPUT

```
-- INNER JOIN Result:
EMP_ID  EMP_NAME      SALARY   DEPT_NAME             LOCATION
------  ------------  -------  --------------------  ----------
101     Ravi Kumar    65000    Information Technology  Hyderabad
102     Priya Sharma  52000    Human Resources         Bangalore
103     Arjun Nair    75000    Information Technology  Hyderabad
104     Sneha Rao     80000    Finance                 Mumbai
105     Kiran Patel   70000    Information Technology  Hyderabad

-- LEFT JOIN Result:
DEPT_ID  DEPT_NAME             EMP_NAME
-------  --------------------  ------------
10       Information Technology  Ravi Kumar
10       Information Technology  Arjun Nair
10       Information Technology  Kiran Patel
20       Human Resources         Priya Sharma
30       Finance                 Sneha Rao
40       Research & Dev          No Employees

-- After DELETE CASCADE (DEPT 10 deleted):
EMP_ID  EMP_NAME      SALARY   DEPT_ID
------  ------------  -------  -------
102     Priya Sharma  52000    20
104     Sneha Rao     80000    30
```

7. PL/SQL Program to Find Factorial of a Number

PL/SQL CODE

```
DECLARE
   v_n    NUMBER := 6;   -- Input: Change this value to test
   v_fact NUMBER := 1;
   i      NUMBER;
BEGIN
   IF v_n < 0 THEN
      DBMS_OUTPUT.PUT_LINE('ERROR: Factorial is not defined for negative numbers.');
   ELSIF v_n = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Factorial of 0 = 1');
   ELSE
      FOR i IN 1..v_n LOOP
         v_fact := v_fact * i;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('Factorial of ' || v_n || ' = ' || v_fact);
   END IF;
END;
/

-- Test with 0
DECLARE
   v_n    NUMBER := 0;
   v_fact NUMBER := 1;
BEGIN
   IF v_n = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Factorial of 0 = 1  (Special Case)');
   ELSE
      FOR i IN 1..v_n LOOP
         v_fact := v_fact * i;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('Factorial of ' || v_n || ' = ' || v_fact);
   END IF;
END;
/
```

OUTPUT

Factorial of 6 = 720

Factorial of 0 = 1 (Special Case)

PL/SQL procedure successfully completed.

8. PL/SQL Program to Check Whether a Number is Palindrome

PL/SQL CODE

```
DECLARE
   v_num   NUMBER := 12321;  -- Test number (change to test others)
   v_temp  NUMBER;
   v_rev   NUMBER := 0;
   v_digit NUMBER;
BEGIN
   v_temp := v_num;

   -- Reverse the number
   WHILE v_temp > 0 LOOP
      v_digit := MOD(v_temp, 10);
      v_rev   := v_rev * 10 + v_digit;
      v_temp  := TRUNC(v_temp / 10);
   END LOOP;

   DBMS_OUTPUT.PUT_LINE('Original Number  : ' || v_num);
   DBMS_OUTPUT.PUT_LINE('Reversed Number  : ' || v_rev);

   -- Check palindrome
   IF v_rev = v_num THEN
      DBMS_OUTPUT.PUT_LINE(v_num || ' is a PALINDROME number.');
   ELSE
      DBMS_OUTPUT.PUT_LINE(v_num || ' is NOT a PALINDROME number.');
   END IF;
END;
/

-- Test with non-palindrome
DECLARE
   v_num   NUMBER := 12345;
   v_temp  NUMBER;
   v_rev   NUMBER := 0;
   v_digit NUMBER;
BEGIN
   v_temp := v_num;
   WHILE v_temp > 0 LOOP
      v_digit := MOD(v_temp, 10);
      v_rev   := v_rev * 10 + v_digit;
      v_temp  := TRUNC(v_temp / 10);
   END LOOP;
   IF v_rev = v_num THEN
      DBMS_OUTPUT.PUT_LINE(v_num || ' is a PALINDROME number.');
   ELSE
      DBMS_OUTPUT.PUT_LINE(v_num || ' is NOT a PALINDROME number.');
   END IF;
END;
/
```

OUTPUT

```
Original Number  : 12321
Reversed Number  : 12321
12321 is a PALINDROME number.

12345 is NOT a PALINDROME number.

PL/SQL procedure successfully completed.
```

9. Demonstration of Predefined Exceptions in PL/SQL

PL/SQL CODE

```
-- Setup
CREATE TABLE SAMPLE_DATA (
   S_ID    NUMBER PRIMARY KEY,
   S_NAME  VARCHAR2(30)
);
INSERT INTO SAMPLE_DATA VALUES (1, 'Alpha');
INSERT INTO SAMPLE_DATA VALUES (2, 'Beta');
INSERT INTO SAMPLE_DATA VALUES (3, 'Gamma');
COMMIT;

-- 1. NO_DATA_FOUND
DECLARE
   v_name SAMPLE_DATA.S_NAME%TYPE;
BEGIN
   SELECT S_NAME INTO v_name FROM SAMPLE_DATA WHERE S_ID = 99;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('[NO_DATA_FOUND] No record exists for S_ID = 99.');
END;
/
-- 2. TOO_MANY_ROWS
DECLARE
   v_name SAMPLE_DATA.S_NAME%TYPE;
BEGIN
   SELECT S_NAME INTO v_name FROM SAMPLE_DATA; -- Multiple rows
EXCEPTION
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('[TOO_MANY_ROWS] Query returned multiple rows unexpectedly.');
END;
/
-- 3. ZERO_DIVIDE
DECLARE
   v_a NUMBER := 50;
   v_b NUMBER := 0;
   v_r NUMBER;
BEGIN
   v_r := v_a / v_b;
   DBMS_OUTPUT.PUT_LINE('Result: ' || v_r);
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('[ZERO_DIVIDE] Cannot divide ' || v_a || ' by zero.');
END;
/
-- 4. DUP_VAL_ON_INDEX
BEGIN
   INSERT INTO SAMPLE_DATA VALUES (1, 'Delta'); -- S_ID=1 already exists
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('[DUP_VAL_ON_INDEX] Primary key 1 already exists.');
END;
/
-- 5. VALUE_ERROR
DECLARE
   v_x VARCHAR2(3);
BEGIN
   v_x := 'ABCDEFG';  -- Too large for VARCHAR2(3)
EXCEPTION
   WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('[VALUE_ERROR] Value is too large for the variable.');
END;
/
```

OUTPUT

[NO_DATA_FOUND] No record exists for S_ID = 99.

[TOO_MANY_ROWS] Query returned multiple rows unexpectedly.

[ZERO_DIVIDE] Cannot divide 50 by zero.

[DUP_VAL_ON_INDEX] Primary key 1 already exists.

[VALUE_ERROR] Value is too large for the variable.

PL/SQL procedure successfully completed.

10. PL/SQL Program using User-Defined Exception for Divide Operation

PL/SQL CODE

```
-- Full Program with User-Defined Exceptions
DECLARE
   v_numerator   NUMBER := 100;
   v_denominator NUMBER := 0;   -- Change to test different scenarios
   v_result      NUMBER;

   -- User-defined exception declarations
   e_divide_by_zero  EXCEPTION;
   e_negative_input  EXCEPTION;
   e_overflow        EXCEPTION;
   c_max_result      CONSTANT NUMBER := 10000;

BEGIN
   -- Validation: Check for negative inputs
   IF v_numerator < 0 OR v_denominator < 0 THEN
      RAISE e_negative_input;
   END IF;
   -- Validation: Check for division by zero
   IF v_denominator = 0 THEN
      RAISE e_divide_by_zero;
   END IF;

   -- Perform Division
   v_result := v_numerator / v_denominator;

   -- Validation: Check overflow
   IF v_result > c_max_result THEN
      RAISE e_overflow;
   END IF;

   DBMS_OUTPUT.PUT_LINE('Division Result: ' || v_numerator ||
                        ' / ' || v_denominator || ' = ' || v_result);
EXCEPTION
   WHEN e_divide_by_zero THEN
      DBMS_OUTPUT.PUT_LINE('USER EXCEPTION: Division by zero is not allowed!');
      DBMS_OUTPUT.PUT_LINE('Denominator must be a non-zero value.');

   WHEN e_negative_input THEN
      DBMS_OUTPUT.PUT_LINE('USER EXCEPTION: Negative inputs are not accepted!');
      DBMS_OUTPUT.PUT_LINE('Both numerator and denominator must be positive.');

   WHEN e_overflow THEN
      DBMS_OUTPUT.PUT_LINE('USER EXCEPTION: Result exceeds maximum limit of '
                           || c_max_result || '!');

   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END;
/
```

OUTPUT

```
-- Test 1: v_numerator=100, v_denominator=0
USER EXCEPTION: Division by zero is not allowed!
Denominator must be a non-zero value.
-- Test 2: v_numerator=100, v_denominator=4
Division Result: 100 / 4 = 25
-- Test 3: v_numerator=-50, v_denominator=5
USER EXCEPTION: Negative inputs are not accepted!
Both numerator and denominator must be positive.
PL/SQL procedure successfully completed.
```

11.PL/SQL Program for Library Book Issue System with Exception Handling

PL/SQL CODE

```
-- Table Definitions
CREATE TABLE LIB_MEMBERS (
   MEM_ID     NUMBER PRIMARY KEY,
   MEM_NAME   VARCHAR2(50),
   ISSUED_CNT NUMBER DEFAULT 0
);

CREATE TABLE LIB_BOOKS (
   BOOK_ID    NUMBER PRIMARY KEY,
   TITLE      VARCHAR2(100),
   AUTHOR     VARCHAR2(50),
   AVAIL_COPY NUMBER
);

CREATE TABLE BOOK_ISSUES (
   ISSUE_ID   NUMBER PRIMARY KEY,
   MEM_ID     NUMBER,
   BOOK_ID    NUMBER,
   ISSUE_DATE DATE DEFAULT SYSDATE,
   DUE_DATE   DATE DEFAULT SYSDATE + 14
);

-- Issue Requests Table (simulates incoming requests)
CREATE TABLE ISSUE_REQUESTS (
   REQ_ID  NUMBER PRIMARY KEY,
   MEM_ID  NUMBER,
   BOOK_ID NUMBER
);

CREATE SEQUENCE iss_seq START WITH 101 INCREMENT BY 1;

-- Insert Sample Data
INSERT INTO LIB_MEMBERS VALUES (1, 'Ravi Kumar',  0);
INSERT INTO LIB_MEMBERS VALUES (2, 'Priya Devi',  0);
INSERT INTO LIB_MEMBERS VALUES (3, 'Arjun Singh', 3); -- Already at limit

INSERT INTO LIB_BOOKS VALUES (201, 'Database Systems',    'Korth',    2);
INSERT INTO LIB_BOOKS VALUES (202, 'Operating Systems',   'Silberschatz', 0); -- No copies
INSERT INTO LIB_BOOKS VALUES (203, 'Data Structures',     'Cormen',   5);

INSERT INTO ISSUE_REQUESTS VALUES (1, 1, 201); -- Valid request
INSERT INTO ISSUE_REQUESTS VALUES (2, 2, 202); -- Book unavailable
INSERT INTO ISSUE_REQUESTS VALUES (3, 3, 203); -- Member at max limit
INSERT INTO ISSUE_REQUESTS VALUES (4, 9, 201); -- Invalid member
COMMIT;

-- Main PL/SQL Block
DECLARE
   v_mem_id   LIB_MEMBERS.MEM_ID%TYPE;
   v_book_id  LIB_BOOKS.BOOK_ID%TYPE;
   v_mem_name LIB_MEMBERS.MEM_NAME%TYPE;
   v_title    LIB_BOOKS.TITLE%TYPE;
   v_avail    LIB_BOOKS.AVAIL_COPY%TYPE;
   v_cnt      LIB_MEMBERS.ISSUED_CNT%TYPE;

   e_book_not_available  EXCEPTION;
   e_max_limit           EXCEPTION;

   CURSOR c_req IS
      SELECT REQ_ID, MEM_ID, BOOK_ID FROM ISSUE_REQUESTS;
```

```
    v_req_id NUMBER;
BEGIN
    OPEN c_req;
    DBMS_OUTPUT.PUT_LINE('===== LIBRARY BOOK ISSUE PROCESSING =====');
    LOOP
        FETCH c_req INTO v_req_id, v_mem_id, v_book_id;
        EXIT WHEN c_req%NOTFOUND;
        BEGIN
            -- Fetch Member
            SELECT MEM_NAME, ISSUED_CNT
            INTO   v_mem_name, v_cnt
            FROM   LIB_MEMBERS WHERE MEM_ID = v_mem_id;

            -- Fetch Book
            SELECT TITLE, AVAIL_COPY
            INTO   v_title, v_avail
            FROM   LIB_BOOKS WHERE BOOK_ID = v_book_id;

            IF v_avail = 0 THEN
                RAISE e_book_not_available;
            END IF;
            IF v_cnt >= 3 THEN
                RAISE e_max_limit;
            END IF;

            -- Issue the book
            INSERT INTO BOOK_ISSUES(ISSUE_ID, MEM_ID, BOOK_ID)
            VALUES (iss_seq.NEXTVAL, v_mem_id, v_book_id);

            UPDATE LIB_BOOKS SET AVAIL_COPY = AVAIL_COPY - 1
            WHERE  BOOK_ID = v_book_id;

            UPDATE LIB_MEMBERS SET ISSUED_CNT = ISSUED_CNT + 1
            WHERE  MEM_ID = v_mem_id;

            DBMS_OUTPUT.PUT_LINE('[SUCCESS] Book ''' || v_title ||
                ''' issued to ' || v_mem_name);
            COMMIT;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('[ERROR] Req#' || v_req_id ||
                    ': Member or Book not found.');
            WHEN e_book_not_available THEN
                DBMS_OUTPUT.PUT_LINE('[ERROR] Req#' || v_req_id ||
                    ': Book ''' || v_title || ''' not available.');
            WHEN e_max_limit THEN
                DBMS_OUTPUT.PUT_LINE('[ERROR] Req#' || v_req_id ||
                    ': ' || v_mem_name || ' has reached maximum issue limit (3).');
        END;
    END LOOP;
    CLOSE c_req;
    DBMS_OUTPUT.PUT_LINE('===== PROCESSING COMPLETE =====');
END;
/
```

OUTPUT

```
===== LIBRARY BOOK ISSUE PROCESSING =====
[SUCCESS] Book "Database Systems" issued to Ravi Kumar
[ERROR] Req#2: Book "Operating Systems" not available.
[ERROR] Req#3: Arjun Singh has reached maximum issue limit (3).
[ERROR] Req#4: Member or Book not found.
===== PROCESSING COMPLETE =====

PL/SQL procedure successfully completed.
```

12. **Practical Usage of Predefined Oracle Packages in PL/SQL**

PL/SQL CODE

```
-- 1. DBMS_OUTPUT Package
BEGIN
   DBMS_OUTPUT.ENABLE(1000000);   -- Enable buffer (1 MB)
   DBMS_OUTPUT.PUT_LINE('=== DBMS_OUTPUT DEMO ===');
   DBMS_OUTPUT.PUT('Hello ');     -- No newline
   DBMS_OUTPUT.PUT_LINE('World!');
   DBMS_OUTPUT.NEW_LINE;          -- Blank line
   DBMS_OUTPUT.PUT_LINE('Buffer size: 1,000,000 bytes');
END;
/

-- 2. DBMS_RANDOM Package
DECLARE
   v_float   NUMBER;
   v_int     NUMBER;
   v_str     VARCHAR2(20);
BEGIN
   DBMS_OUTPUT.PUT_LINE('=== DBMS_RANDOM DEMO ===');

   -- Random float between 0 and 1
   v_float := DBMS_RANDOM.VALUE;
   DBMS_OUTPUT.PUT_LINE('Random Float (0-1)    : ' || ROUND(v_float, 6));

   -- Random integer between 1 and 100
   v_int := TRUNC(DBMS_RANDOM.VALUE(1, 100));
   DBMS_OUTPUT.PUT_LINE('Random Integer (1-100): ' || v_int);

   -- Random uppercase string of length 8
   v_str := DBMS_RANDOM.STRING('U', 8);
   DBMS_OUTPUT.PUT_LINE('Random String (U,8)   : ' || v_str);

   -- Random lowercase string of length 6
   v_str := DBMS_RANDOM.STRING('L', 6);
   DBMS_OUTPUT.PUT_LINE('Random String (L,6)   : ' || v_str);

   DBMS_RANDOM.TERMINATE;
END;
/

-- 3. Date Functions and SYSDATE
DECLARE
   v_today     DATE := SYSDATE;
   v_future    DATE;
   v_months    NUMBER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('=== DATE FUNCTION DEMO ===');
   DBMS_OUTPUT.PUT_LINE('Today          : ' ||
      TO_CHAR(v_today, 'DD-MON-YYYY HH24:MI:SS'));
   DBMS_OUTPUT.PUT_LINE('Day of Week    : ' ||
      TO_CHAR(v_today, 'DAY'));
   DBMS_OUTPUT.PUT_LINE('Day of Year    : ' ||
      TO_CHAR(v_today, 'DDD'));

   -- Add 3 months
   v_future := ADD_MONTHS(v_today, 3);
   DBMS_OUTPUT.PUT_LINE('+3 Months      : ' ||
      TO_CHAR(v_future, 'DD-MON-YYYY'));

   -- Months between two dates
   v_months := MONTHS_BETWEEN(v_future, v_today);
```

```
    DBMS_OUTPUT.PUT_LINE('Months Between : ' || v_months);

    -- Last day of month
    DBMS_OUTPUT.PUT_LINE('Last of Month  : ' ||
        TO_CHAR(LAST_DAY(v_today), 'DD-MON-YYYY'));
END;
/

-- 4. DBMS_UTILITY – Performance Timing
DECLARE
    v_start  NUMBER;
    v_end    NUMBER;
    v_elapsed NUMBER;
BEGIN
    v_start := DBMS_UTILITY.GET_TIME;

    -- Simulate workload
    FOR i IN 1..100000 LOOP
        NULL;
    END LOOP;

    v_end     := DBMS_UTILITY.GET_TIME;
    v_elapsed := (v_end - v_start) / 100;  -- in seconds
    DBMS_OUTPUT.PUT_LINE('=== DBMS_UTILITY DEMO ===');
    DBMS_OUTPUT.PUT_LINE('Elapsed Time: ' || v_elapsed || ' seconds');
END;
/
```

OUTPUT

```
=== DBMS_OUTPUT DEMO ===
Hello World!

Buffer size: 1,000,000 bytes

=== DBMS_RANDOM DEMO ===
Random Float (0-1)    : 0.748213
Random Integer (1-100): 63
Random String (U,8)   : KXRPMNBQ
Random String (L,6)   : fvzjqr

=== DATE FUNCTION DEMO ===
Today          : 01-JAN-2025 10:30:45
Day of Week    : WEDNESDAY
Day of Year    : 001
+3 Months      : 01-APR-2025
Months Between : 3
Last of Month  : 31-JAN-2025

=== DBMS_UTILITY DEMO ===
Elapsed Time: 0.02 seconds
```

13.Create and Manage Object Types, Tables, and Methods using Oracle SQL

SQL / PL-SQL CODE

```
-- Step 1: Create Object Type Specification
CREATE OR REPLACE TYPE STUDENT_TYPE AS OBJECT (
   ROLL_NO   NUMBER,
   STU_NAME  VARCHAR2(50),
   MARKS     NUMBER(5,2),
   -- Member Function Declaration
   MEMBER FUNCTION GET_GRADE RETURN VARCHAR2,
   MEMBER FUNCTION GET_DETAILS RETURN VARCHAR2
);
/

-- Step 2: Create Object Type Body (Implementation)
CREATE OR REPLACE TYPE BODY STUDENT_TYPE AS

   MEMBER FUNCTION GET_GRADE RETURN VARCHAR2 IS
      v_grade VARCHAR2(5);
   BEGIN
      IF    SELF.MARKS >= 90 THEN v_grade := 'O';
      ELSIF SELF.MARKS >= 75 THEN v_grade := 'A+';
      ELSIF SELF.MARKS >= 60 THEN v_grade := 'A';
      ELSIF SELF.MARKS >= 50 THEN v_grade := 'B';
      ELSIF SELF.MARKS >= 40 THEN v_grade := 'C';
      ELSE                        v_grade := 'F';
      END IF;
      RETURN v_grade;
   END GET_GRADE;

   MEMBER FUNCTION GET_DETAILS RETURN VARCHAR2 IS
   BEGIN
      RETURN 'Roll: ' || SELF.ROLL_NO ||
           ' | Name: ' || SELF.STU_NAME ||
           ' | Marks: ' || SELF.MARKS ||
           ' | Grade: ' || SELF.GET_GRADE();
   END GET_DETAILS;

END;
/

-- Step 3: Create Object Table
CREATE TABLE STUDENT_OBJ_TABLE OF STUDENT_TYPE (
   PRIMARY KEY (ROLL_NO)
);

-- Step 4: Insert Object Instances
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(101, 'Arjun Reddy',  92.5));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(102, 'Priya Devi',   78.0));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(103, 'Kiran Babu',   55.5));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(104, 'Divya Nair',   38.0));
COMMIT;

-- Step 5: Query Object Table
SELECT S.ROLL_NO, S.STU_NAME, S.MARKS, S.GET_GRADE() AS GRADE
FROM   STUDENT_OBJ_TABLE S
ORDER  BY S.ROLL_NO;

-- Step 6: Call Member Method GET_DETAILS
DECLARE
   v_stu STUDENT_TYPE;
BEGIN
   SELECT VALUE(S) INTO v_stu
```

```
    FROM   STUDENT_OBJ_TABLE S
    WHERE  S.ROLL_NO = 101;

    DBMS_OUTPUT.PUT_LINE(v_stu.GET_DETAILS());
END;
/

-- Step 7: Nested Object Type - ADDRESS_TYPE
CREATE OR REPLACE TYPE ADDRESS_TYPE AS OBJECT (
    STREET   VARCHAR2(50),
    CITY     VARCHAR2(30),
    PINCODE  VARCHAR2(10)
);
/

CREATE OR REPLACE TYPE EMP_OBJ AS OBJECT (
    EMP_ID   NUMBER,
    EMP_NAME VARCHAR2(50),
    ADDR     ADDRESS_TYPE   -- Nested Object
);
/

CREATE TABLE EMP_OBJ_TABLE OF EMP_OBJ;

INSERT INTO EMP_OBJ_TABLE VALUES (
    EMP_OBJ(1001, 'Ravi Kumar',
       ADDRESS_TYPE('12 MG Road', 'Hyderabad', '500001'))
);
COMMIT;

SELECT E.EMP_ID, E.EMP_NAME,
      E.ADDR.STREET, E.ADDR.CITY, E.ADDR.PINCODE
FROM   EMP_OBJ_TABLE E;
```

OUTPUT

```
Object type STUDENT_TYPE created.
Object type body STUDENT_TYPE created.
Table STUDENT_OBJ_TABLE created.
4 rows inserted.

ROLL_NO  STU_NAME     MARKS  GRADE
-------  ------------ -----  -----
101      Arjun Reddy   92.5   O
102      Priya Devi    78.0   A+
103      Kiran Babu    55.5   B
104      Divya Nair    38.0   F

Roll: 101 | Name: Arjun Reddy | Marks: 92.5 | Grade: O

EMP_ID  EMP_NAME    STREET      CITY        PINCODE
------  ----------  ----------  ----------  -------
1001    Ravi Kumar  12 MG Road  Hyderabad   500001
```

Case Study-1

14. Selection, Projection, and Sorting Queries on a Simple Table

SQL CODE

```
-- Create Table
CREATE TABLE STUDENT (
   ROLL_NO NUMBER PRIMARY KEY,
   S_NAME  VARCHAR2(50),
   BRANCH  VARCHAR2(20),
   CITY    VARCHAR2(30),
   CGPA    NUMBER(3,1)
);

-- Insert Records
INSERT INTO STUDENT VALUES (1,  'Arjun Reddy',     'CSE', 'Hyderabad', 9.2);
INSERT INTO STUDENT VALUES (2,  'Priya Sharma',    'ECE', 'Bangalore', 8.5);
INSERT INTO STUDENT VALUES (3,  'Kiran Babu',      'CSE', 'Chennai',   7.8);
INSERT INTO STUDENT VALUES (4,  'Divya Nair',      'IT',  'Hyderabad', 8.1);
INSERT INTO STUDENT VALUES (5,  'Suresh Goud',     'CSE', 'Mumbai',    6.5);
INSERT INTO STUDENT VALUES (6,  'Meena Kumari',    'ECE', 'Pune',      9.0);
INSERT INTO STUDENT VALUES (7,  'Vikram Singh',    'IT',  'Delhi',     7.2);
INSERT INTO STUDENT VALUES (8,  'Anjali Patel',    'CSE', 'Hyderabad', 8.9);
INSERT INTO STUDENT VALUES (9,  'Ravi Kumar',      'MECH','Chennai',   7.5);
INSERT INTO STUDENT VALUES (10, 'Deepa Menon',     'IT',  'Bangalore', 9.1);
COMMIT;

-- 1. PROJECTION: Select only Name and CGPA
SELECT S_NAME, CGPA FROM STUDENT;

-- 2. SELECTION: Filter CSE students
SELECT * FROM STUDENT WHERE BRANCH = 'CSE';

-- 3. Selection + Projection: Name & CGPA where CGPA > 8
SELECT S_NAME, BRANCH, CGPA
FROM   STUDENT
WHERE  CGPA > 8.0;

-- 4. SORTING: Order by CGPA Descending
SELECT ROLL_NO, S_NAME, BRANCH, CGPA
FROM   STUDENT
ORDER  BY CGPA DESC;

-- 5. Selection + Projection + Sorting (Combined)
SELECT ROLL_NO, S_NAME, CITY, CGPA
FROM   STUDENT
WHERE  BRANCH IN ('CSE', 'IT')
ORDER  BY CGPA DESC;

-- 6. DISTINCT – Unique cities
SELECT DISTINCT CITY FROM STUDENT ORDER BY CITY;

-- 7. BETWEEN – Students with CGPA between 7.5 and 8.5
SELECT S_NAME, CGPA FROM STUDENT
WHERE  CGPA BETWEEN 7.5 AND 8.5;

-- 8. LIKE – Students whose name starts with 'A'
SELECT S_NAME, BRANCH FROM STUDENT
WHERE  S_NAME LIKE 'A%';
```

OUTPUT

```
-- 3. Selection + Projection (CGPA > 8):
S_NAME         BRANCH  CGPA
-----------    ------  ----
Arjun Reddy    CSE     9.2
Priya Sharma   ECE     8.5
Divya Nair     IT      8.1
Meena Kumari   ECE     9.0
Anjali Patel   CSE     8.9
Deepa Menon    IT      9.1

-- 4. Sorting by CGPA DESC:
ROLL_NO  S_NAME         BRANCH  CGPA
-------  -------------  ------  ----
1        Arjun Reddy    CSE     9.2
10       Deepa Menon    IT      9.1
6        Meena Kumari   ECE     9.0
8        Anjali Patel   CSE     8.9
...

-- 6. DISTINCT Cities:
CITY
---------
Bangalore
Chennai
Delhi
Hyderabad
Mumbai
Pune
```

Case Study-2

15. ER Diagram for a Company Database

SQL CODE

```
-- ER Diagram Translation to SQL Tables

-- 1. DEPARTMENT Table (Created first for FK reference)
CREATE TABLE COMPANY_DEPT (
   DEPT_NO   NUMBER      PRIMARY KEY,
   DEPT_NAME VARCHAR2(40) NOT NULL,
   LOCATION  VARCHAR2(30),
   MGR_ID    NUMBER       -- FK to COMPANY_EMP (added after)
);

-- 2. EMPLOYEE Table
CREATE TABLE COMPANY_EMP (
   EMP_ID    NUMBER      PRIMARY KEY,
   EMP_NAME  VARCHAR2(50) NOT NULL,
   DOB       DATE,
   SALARY    NUMBER(10,2),
   DEPT_NO   NUMBER,
   CONSTRAINT fk_emp_dept FOREIGN KEY (DEPT_NO)
      REFERENCES COMPANY_DEPT(DEPT_NO)
);

-- Add Manager FK to DEPARTMENT
ALTER TABLE COMPANY_DEPT
   ADD CONSTRAINT fk_dept_mgr FOREIGN KEY (MGR_ID)
   REFERENCES COMPANY_EMP(EMP_ID);

-- 3. PROJECT Table
CREATE TABLE COMPANY_PROJ (
   PROJ_ID   NUMBER      PRIMARY KEY,
   PROJ_NAME VARCHAR2(60) NOT NULL,
   BUDGET    NUMBER(12,2),
   DEPT_NO   NUMBER,
   CONSTRAINT fk_proj_dept FOREIGN KEY (DEPT_NO)
      REFERENCES COMPANY_DEPT(DEPT_NO)
);

-- 4. WORKS_ON Table (M:N Relationship)
CREATE TABLE WORKS_ON (
   EMP_ID  NUMBER,
   PROJ_ID NUMBER,
   HOURS   NUMBER(5,2),
   CONSTRAINT pk_works_on PRIMARY KEY (EMP_ID, PROJ_ID),
   CONSTRAINT fk_wo_emp  FOREIGN KEY (EMP_ID)  REFERENCES COMPANY_EMP(EMP_ID),
   CONSTRAINT fk_wo_proj FOREIGN KEY (PROJ_ID) REFERENCES COMPANY_PROJ(PROJ_ID)
);

-- 5. DEPENDENT (Weak Entity)
CREATE TABLE DEPENDENT (
   EMP_ID    NUMBER,
   DEP_NAME  VARCHAR2(40),
   RELATION  VARCHAR2(20),
   CONSTRAINT pk_dep PRIMARY KEY (EMP_ID, DEP_NAME),
   CONSTRAINT fk_dep_emp FOREIGN KEY (EMP_ID) REFERENCES COMPANY_EMP(EMP_ID)
);

-- Insert Sample Data
INSERT INTO COMPANY_DEPT(DEPT_NO,DEPT_NAME,LOCATION) VALUES (10,'IT','Hyderabad');
```

```
INSERT INTO COMPANY_DEPT(DEPT_NO,DEPT_NAME,LOCATION) VALUES (20,'HR','Bangalore');

INSERT INTO COMPANY_EMP VALUES (101,'Ravi Kumar',  DATE'1990-05-15', 70000, 10);
INSERT INTO COMPANY_EMP VALUES (102,'Priya Devi',  DATE'1992-08-20', 55000, 20);
INSERT INTO COMPANY_EMP VALUES (103,'Arjun Nair',  DATE'1988-03-10', 80000, 10);

UPDATE COMPANY_DEPT SET MGR_ID = 101 WHERE DEPT_NO = 10;
UPDATE COMPANY_DEPT SET MGR_ID = 102 WHERE DEPT_NO = 20;

INSERT INTO COMPANY_PROJ VALUES (501,'ERP System',  500000, 10);
INSERT INTO COMPANY_PROJ VALUES (502,'HR Portal',   250000, 20);

INSERT INTO WORKS_ON VALUES (101, 501, 30.0);
INSERT INTO WORKS_ON VALUES (103, 501, 25.5);
INSERT INTO WORKS_ON VALUES (101, 502, 10.0);

INSERT INTO DEPENDENT VALUES (101, 'Kavya',   'Spouse');
INSERT INTO DEPENDENT VALUES (101, 'Rahul',   'Son');
COMMIT;

-- Comprehensive Query: Employee, Department, Project
SELECT E.EMP_ID, E.EMP_NAME, D.DEPT_NAME, P.PROJ_NAME, W.HOURS
FROM   COMPANY_EMP  E
JOIN   COMPANY_DEPT D ON E.DEPT_NO  = D.DEPT_NO
JOIN   WORKS_ON     W ON E.EMP_ID   = W.EMP_ID
JOIN   COMPANY_PROJ P ON W.PROJ_ID  = P.PROJ_ID
ORDER  BY E.EMP_ID;
```

OUTPUT

```
Tables created successfully.
Data inserted and committed.

-- Final JOIN Query Output:
EMP_ID  EMP_NAME    DEPT_NAME  PROJ_NAME   HOURS
------  ----------  ---------  ----------  -----
101     Ravi Kumar  IT         ERP System  30.0
101     Ravi Kumar  IT         HR Portal   10.0
103     Arjun Nair  IT         ERP System  25.5

-- ER Diagram Summary:
Entities   : EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT
Relationships: WORKS_IN (M:1), MANAGES (1:1),
               WORKS_ON (M:N), HAS DEPENDENT (1:N),
               CONTROLS (1:N)
Cardinalities enforced via FK constraints.
```