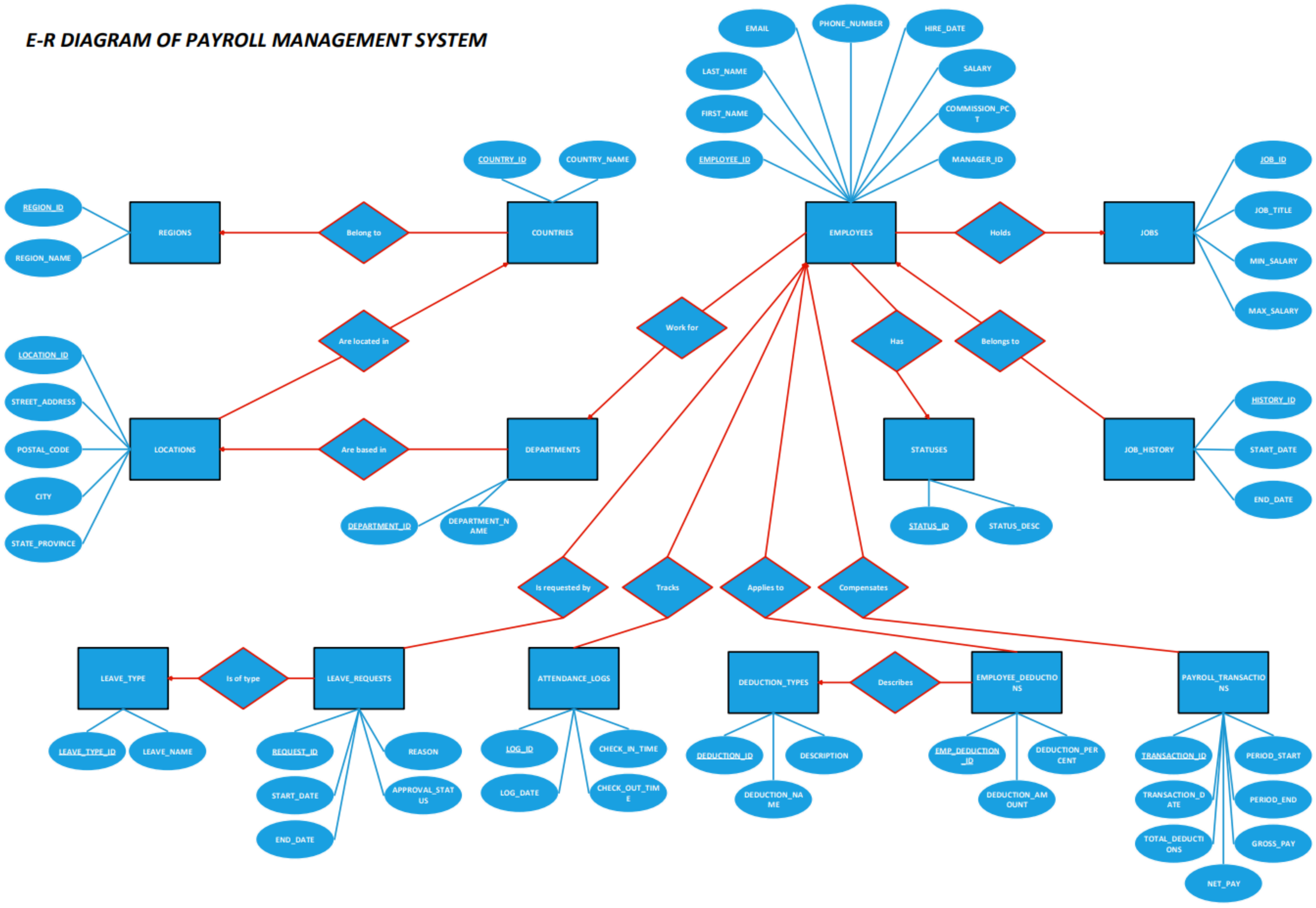
COMP305 DMBS - Final Project

Payroll Management System

Instructor:
Prof.Dr. Adem Karahoca

Group Members:
Kerem Ataç - 042201043
Ulaş Uzun - 042301078
Arda Tuna Küpçü - 042301093
Nihat Eren Şen - 042101099

E-R DIAGRAM OF PAYROLL MANAGEMENT SYSTEM

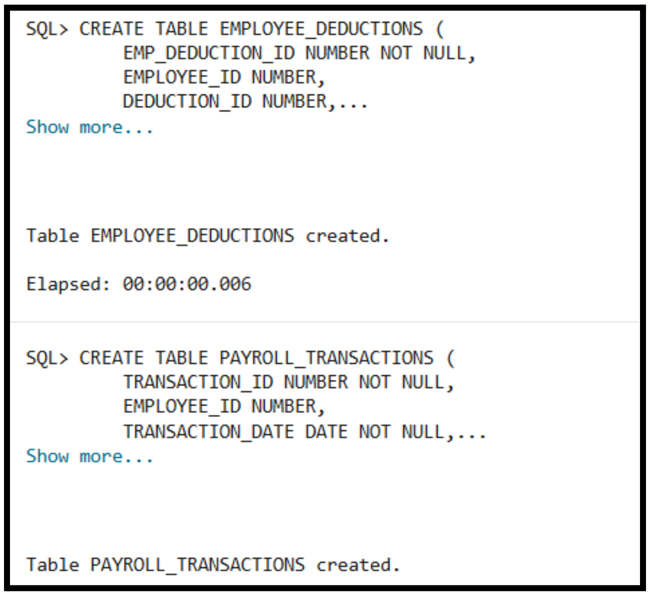


2. Table Creation Codes

```
-- 1. REGIONS Table
CREATE TABLE REGIONS (
    REGION_ID NUMBER NOT NULL,
    REGION_NAME VARCHAR2(50)
);

-- 2. COUNTRIES Table
CREATE TABLE COUNTRIES (
    COUNTRY_ID CHAR(2) NOT NULL,
    COUNTRY_NAME VARCHAR2(60),
    REGION_ID NUMBER
);

-- 3. LOCATIONS Table
CREATE TABLE LOCATIONS (
    LOCATION_ID NUMBER NOT NULL,
    STREET_ADDRESS VARCHAR2(100),
    POSTAL_CODE VARCHAR2(20),
    CITY VARCHAR2(50) NOT NULL,
    STATE_PROVINCE VARCHAR2(50),
    COUNTRY_ID CHAR(2)
);

-- 4. DEPARTMENTS Table
CREATE TABLE DEPARTMENTS (
    DEPARTMENT_ID NUMBER NOT NULL,
    DEPARTMENT_NAME VARCHAR2(50) NOT NULL,
    LOCATION_ID NUMBER
);

-- 5. JOBS Table
CREATE TABLE JOBS (
    JOB_ID VARCHAR2(20) NOT NULL,
    JOB_TITLE VARCHAR2(50) NOT NULL,
    MIN_SALARY NUMBER(10, 2),
    MAX_SALARY NUMBER(10, 2)
);

-- 6. STATUSES Table
CREATE TABLE STATUSES (
    STATUS_ID NUMBER NOT NULL,
    STATUS_DESC VARCHAR2(30) -- e.g., 'Active', 'Terminated'
);

-- 7. EMPLOYEES Table
CREATE TABLE EMPLOYEES (
    EMPLOYEE_ID NUMBER NOT NULL,
    FIRST_NAME VARCHAR2(50),
    LAST_NAME VARCHAR2(50) NOT NULL,
    EMAIL VARCHAR2(100),
    PHONE_NUMBER VARCHAR2(20),
    HIRE_DATE DATE NOT NULL,
    JOB_ID VARCHAR2(20),
    SALARY NUMBER(10, 2),
    COMMISSION_PCT NUMBER(3, 2),
    MANAGER_ID NUMBER,
    DEPARTMENT_ID NUMBER,
    STATUS_ID NUMBER
);

-- 8. JOB_HISTORY Table
CREATE TABLE JOB_HISTORY (
    HISTORY_ID NUMBER NOT NULL,
    EMPLOYEE_ID NUMBER NOT NULL,
    START_DATE DATE NOT NULL,
    END_DATE DATE,
    JOB_ID VARCHAR2(20),
    DEPARTMENT_ID NUMBER
);
```

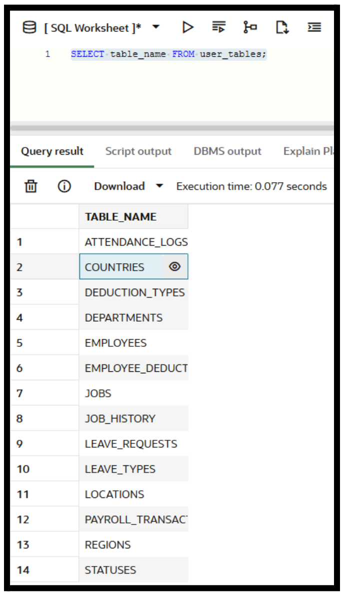
```
-- 9. LEAVE_TYPES Table
CREATE TABLE LEAVE_TYPES (
    LEAVE_TYPE_ID NUMBER NOT NULL,
    LEAVE_NAME VARCHAR2(30) -- e.g., 'Annual', 'Sick',
'Maternity'
);

-- 10. LEAVE_REQUESTS Table
CREATE TABLE LEAVE_REQUESTS (
    REQUEST_ID NUMBER NOT NULL,
    EMPLOYEE_ID NUMBER,
    LEAVE_TYPE_ID NUMBER,
    START_DATE DATE NOT NULL,
    END_DATE DATE NOT NULL,
    REASON VARCHAR2(200),
    APPROVAL_STATUS VARCHAR2(20) -- e.g., 'Pending', 'Approved'
);

-- 11. ATTENDANCE_LOGS Table
CREATE TABLE ATTENDANCE_LOGS (
    LOG_ID NUMBER NOT NULL,
    EMPLOYEE_ID NUMBER,
    LOG_DATE DATE NOT NULL,
    CHECK_IN_TIME TIMESTAMP,
    CHECK_OUT_TIME TIMESTAMP
);

-- 12. DEDUCTION_TYPES Table
CREATE TABLE DEDUCTION_TYPES (
    DEDUCTION_ID NUMBER NOT NULL,
    DEDUCTION_NAME VARCHAR2(50), -- e.g., 'Health Insurance',
'Tax'
    DESCRIPTION VARCHAR2(100)
);
```

```
-- 13. EMPLOYEE_DEDUCTIONS Table
CREATE TABLE EMPLOYEE_DEDUCTIONS (
    EMP_DEDUCTION_ID NUMBER NOT NULL,
    EMPLOYEE_ID NUMBER,
    DEDUCTION_ID NUMBER,
    DEDUCTION_AMOUNT NUMBER(10, 2), -- Fixed amount if
applicable
    DEDUCTION_PERCENT NUMBER(4, 2)  -- Percentage if applicable
);

-- 14. PAYROLL_TRANSACTIONS Table
CREATE TABLE PAYROLL_TRANSACTIONS (
    TRANSACTION_ID NUMBER NOT NULL,
    EMPLOYEE_ID NUMBER,
    TRANSACTION_DATE DATE NOT NULL,
    PERIOD_START DATE,
    PERIOD_END DATE,
    GROSS_PAY NUMBER(10, 2),
    TOTAL_DEDUCTIONS NUMBER(10, 2),
    NET_PAY NUMBER(10, 2)
);
```



```
SQL> CREATE TABLE EMPLOYEE_DEDUCTIONS (
         EMP_DEDUCTION_ID NUMBER NOT NULL,
         EMPLOYEE_ID NUMBER,
         DEDUCTION_ID NUMBER,...
Show more...


Table EMPLOYEE_DEDUCTIONS created.

Elapsed: 00:00:00.006

SQL> CREATE TABLE PAYROLL_TRANSACTIONS (
         TRANSACTION_ID NUMBER NOT NULL,
         EMPLOYEE_ID NUMBER,
         TRANSACTION_DATE DATE NOT NULL,...
Show more...


Table PAYROLL_TRANSACTIONS created.
```

Figure1: Some table creation script outputs



```
SELECT table_name FROM user_tables;
```

Execution time: 0.077 seconds

	TABLE_NAME
1	ATTENDANCE_LOGS
2	COUNTRIES
3	DEDUCTION_TYPES
4	DEPARTMENTS
5	EMPLOYEES
6	EMPLOYEE_DEDUCT
7	JOBS
8	JOB_HISTORY
9	LEAVE_REQUESTS
10	LEAVE_TYPES
11	LOCATIONS
12	PAYROLL_TRANSAC
13	REGIONS
14	STATUSES

Figure2: Table Creation Proof

3. Adding of primary/foreign keys

3.1 Primary Keys Scripts

```
-- Module A: Location, Organization PKs
ALTER TABLE REGIONS ADD CONSTRAINT PK_REGIONS PRIMARY KEY (REGION_ID);
ALTER TABLE COUNTRIES ADD CONSTRAINT PK_COUNTRIES PRIMARY KEY (COUNTRY_ID);
ALTER TABLE LOCATIONS ADD CONSTRAINT PK_LOCATIONS PRIMARY KEY (LOCATION_ID);
ALTER TABLE DEPARTMENTS ADD CONSTRAINT PK_DEPARTMENTS PRIMARY KEY (DEPARTMENT_ID);

-- Module B: HR Core PKs
ALTER TABLE JOBS ADD CONSTRAINT PK_JOBS PRIMARY KEY (JOB_ID);
ALTER TABLE STATUSES ADD CONSTRAINT PK_STATUSES PRIMARY KEY (STATUS_ID);
ALTER TABLE EMPLOYEES ADD CONSTRAINT PK_EMPLOYEES PRIMARY KEY (EMPLOYEE_ID);
ALTER TABLE JOB_HISTORY ADD CONSTRAINT PK_JOB_HISTORY PRIMARY KEY (HISTORY_ID);

-- Module C: Time, Attendance PKs
ALTER TABLE LEAVE_TYPES ADD CONSTRAINT PK_LEAVE_TYPES PRIMARY KEY (LEAVE_TYPE_ID);
ALTER TABLE LEAVE_REQUESTS ADD CONSTRAINT PK_LEAVE_REQUESTS PRIMARY KEY (REQUEST_ID);
ALTER TABLE ATTENDANCE_LOGS ADD CONSTRAINT PK_ATTENDANCE_LOGS PRIMARY KEY (LOG_ID);

-- Module D: Payroll PKs
ALTER TABLE DEDUCTION_TYPES ADD CONSTRAINT PK_DEDUCTION_TYPES PRIMARY KEY
(DEDUCTION_ID);
ALTER TABLE EMPLOYEE_DEDUCTIONS ADD CONSTRAINT PK_EMP_DEDUCTIONS PRIMARY KEY
(EMP_DEDUCTION_ID);
ALTER TABLE PAYROLL_TRANSACTIONS ADD CONSTRAINT PK_PAYROLL_TRANS PRIMARY KEY
(TRANSACTION_ID);
```

3.2 Foreign Keys Scripts

```
-- 1. Link COUNTRIES to REGIONS
ALTER TABLE COUNTRIES ADD CONSTRAINT FK_COUNTRIES_REGION
FOREIGN KEY (REGION_ID) REFERENCES REGIONS(REGION_ID);

-- 2. Link LOCATIONS to COUNTRIES
ALTER TABLE LOCATIONS ADD CONSTRAINT FK_LOCATIONS_COUNTRY
FOREIGN KEY (COUNTRY_ID) REFERENCES COUNTRIES(COUNTRY_ID);

-- 3. Link DEPARTMENTS to LOCATIONS
ALTER TABLE DEPARTMENTS ADD CONSTRAINT FK_DEP_LOCATION
FOREIGN KEY (LOCATION_ID) REFERENCES LOCATIONS(LOCATION_ID);
```

```sql
-- 4. Link EMPLOYEES to JOBS, DEPARTMENTS, and STATUSES
ALTER TABLE EMPLOYEES ADD CONSTRAINT FK_EMP_JOB
FOREIGN KEY (JOB_ID) REFERENCES JOBS(JOB_ID);

ALTER TABLE EMPLOYEES ADD CONSTRAINT FK_EMP_DEP
FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID);

ALTER TABLE EMPLOYEES ADD CONSTRAINT FK_EMP_STATUS
FOREIGN KEY (STATUS_ID) REFERENCES STATUSES(STATUS_ID);

-- Self-Join: Link EMPLOYEES to EMPLOYEES (for Manager)
ALTER TABLE EMPLOYEES ADD CONSTRAINT FK_EMP_MANAGER
FOREIGN KEY (MANAGER_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);

-- 5. Link JOB_HISTORY to EMPLOYEES, JOBS, and DEPARTMENTS
ALTER TABLE JOB_HISTORY ADD CONSTRAINT FK_JH_EMP
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);

ALTER TABLE JOB_HISTORY ADD CONSTRAINT FK_JH_JOB
FOREIGN KEY (JOB_ID) REFERENCES JOBS(JOB_ID);

ALTER TABLE JOB_HISTORY ADD CONSTRAINT FK_JH_DEP
FOREIGN KEY (DEPARTMENT_ID) REFERENCES DEPARTMENTS(DEPARTMENT_ID);

-- 6. Link LEAVE_REQUESTS to EMPLOYEES and LEAVE_TYPES
ALTER TABLE LEAVE_REQUESTS ADD CONSTRAINT FK_LR_EMP
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);

ALTER TABLE LEAVE_REQUESTS ADD CONSTRAINT FK_LR_TYPE
FOREIGN KEY (LEAVE_TYPE_ID) REFERENCES LEAVE_TYPES(LEAVE_TYPE_ID);

-- 7. Link ATTENDANCE_LOGS to EMPLOYEES
ALTER TABLE ATTENDANCE_LOGS ADD CONSTRAINT FK_ATT_EMP
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);

-- 8. Link EMPLOYEE_DEDUCTIONS to EMPLOYEES and DEDUCTION_TYPES
ALTER TABLE EMPLOYEE_DEDUCTIONS ADD CONSTRAINT FK_ED_EMP
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);

ALTER TABLE EMPLOYEE_DEDUCTIONS ADD CONSTRAINT FK_ED_TYPE
FOREIGN KEY (DEDUCTION_ID) REFERENCES DEDUCTION_TYPES(DEDUCTION_ID);

-- 9. Link PAYROLL_TRANSACTIONS to EMPLOYEES
ALTER TABLE PAYROLL_TRANSACTIONS ADD CONSTRAINT FK_PT_EMP
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID);
```

4. Performing Data Entry

4.1 Create the Package Specification

```
CREATE OR REPLACE PACKAGE PAYROLL_ENTRY_PKG AS
    -- Procedure to add a Region
    PROCEDURE ADD_REGION(
        p_region_id IN NUMBER,
        p_region_name IN VARCHAR2
    );

    -- Procedure to add a Job
    PROCEDURE ADD_JOB(
        p_job_id IN VARCHAR2,
        p_job_title IN VARCHAR2,
        p_min_sal IN NUMBER,
        p_max_sal IN NUMBER
    );

    -- Procedure to add a Department
    PROCEDURE ADD_DEPARTMENT(
        p_dept_id IN NUMBER,
        p_dept_name IN VARCHAR2,
        p_loc_id IN NUMBER
    );

    -- Procedure to add an Employee
    PROCEDURE ADD_EMPLOYEE(
        p_emp_id IN NUMBER,
        p_first_name IN VARCHAR2,
        p_last_name IN VARCHAR2,
        p_email IN VARCHAR2,
        p_hire_date IN DATE,
        p_job_id IN VARCHAR2,
        p_salary IN NUMBER,
        p_dept_id IN NUMBER,
        p_status_id IN NUMBER
    );
END PAYROLL_ENTRY_PKG;
/
```

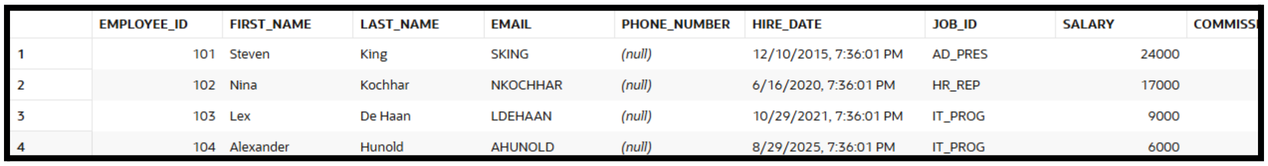
4.2 Create the Package Body

```
CREATE OR REPLACE PACKAGE BODY PAYROLL_ENTRY_PKG AS

    -- 1. Implementation for Region
    PROCEDURE ADD_REGION(p_region_id IN NUMBER, p_region_name IN VARCHAR2) IS
    BEGIN
        INSERT INTO REGIONS (REGION_ID, REGION_NAME)
        VALUES (p_region_id, p_region_name);
        COMMIT;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Region ID ' || p_region_id || ' already
exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error adding region: ' || SQLERRM);
    END ADD_REGION;

    -- 2. Implementation for Job
    PROCEDURE ADD_JOB(p_job_id IN VARCHAR2, p_job_title IN VARCHAR2, p_min_sal IN
NUMBER, p_max_sal IN NUMBER) IS
    BEGIN
        INSERT INTO JOBS (JOB_ID, JOB_TITLE, MIN_SALARY, MAX_SALARY)
        VALUES (p_job_id, p_job_title, p_min_sal, p_max_sal);
        COMMIT;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Job ID ' || p_job_id || ' already exists.');
    END ADD_JOB;

    -- 3. Implementation for Department
    PROCEDURE ADD_DEPARTMENT(p_dept_id IN NUMBER, p_dept_name IN VARCHAR2, p_loc_id IN
NUMBER) IS
    BEGIN
        INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME, LOCATION_ID)
        VALUES (p_dept_id, p_dept_name, p_loc_id);
        COMMIT;
    END ADD_DEPARTMENT;

    -- 4. Implementation for Employee
    PROCEDURE ADD_EMPLOYEE(
        p_emp_id IN NUMBER,
        p_first_name IN VARCHAR2,
        p_last_name IN VARCHAR2,
        p_email IN VARCHAR2,
        p_hire_date IN DATE,
        p_job_id IN VARCHAR2,
        p_salary IN NUMBER,
        p_dept_id IN NUMBER,
        p_status_id IN NUMBER
    ) IS
    BEGIN
        INSERT INTO EMPLOYEES (
            EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE,
            JOB_ID, SALARY, DEPARTMENT_ID, STATUS_ID
        ) VALUES (
            p_emp_id, p_first_name, p_last_name, p_email, p_hire_date,
            p_job_id, p_salary, p_dept_id, p_status_id
        );
        COMMIT;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error adding employee: ' || SQLERRM);
    END ADD_EMPLOYEE;

END PAYROLL_ENTRY_PKG;
/
```

4.3 Calling the Package (Performing Data Entry)

```
SET SERVEROUTPUT ON;

BEGIN
    -- A. Setup Lookup Data (Safe Inserts)

    -- 1. Regions (Handled by Package Exception, but calling here)
    PAYROLL_ENTRY_PKG.ADD_REGION(1, 'Americas');
    PAYROLL_ENTRY_PKG.ADD_REGION(2, 'EMEA');

    -- 2. Countries (Wrapped to ignore duplicates)
    BEGIN
        INSERT INTO COUNTRIES (COUNTRY_ID, COUNTRY_NAME, REGION_ID) VALUES ('US',
'United States', 1);
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL; -- Ignore if exists
    END;

    BEGIN
        INSERT INTO COUNTRIES (COUNTRY_ID, COUNTRY_NAME, REGION_ID) VALUES ('UK',
'United Kingdom', 2);
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    -- 3. Locations
    BEGIN
```

```
        INSERT INTO LOCATIONS VALUES (100, '2004 Charade Rd', '98199', 'Seattle',
'Washington', 'US');
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    BEGIN
        INSERT INTO LOCATIONS VALUES (200, '8204 Arthur St', 'SW1 4RW', 'London',
NULL, 'UK');
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    -- 4. Statuses
    BEGIN
        INSERT INTO STATUSES VALUES (1, 'Active');
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    BEGIN
        INSERT INTO STATUSES VALUES (2, 'On Leave');
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    -- B. Use Package to add Jobs
    -- (The package already has exception handling, so these are safe)
    PAYROLL_ENTRY_PKG.ADD_JOB('IT_PROG', 'Programmer', 4000, 10000);
    PAYROLL_ENTRY_PKG.ADD_JOB('HR_REP', 'HR Representative', 4000, 9000);
    PAYROLL_ENTRY_PKG.ADD_JOB('AD_PRES', 'President', 20000, 40000);

    -- C. Use Package to add Departments
    -- (We will add a quick check inside the calls or just rely on the DB not
crashing)
    BEGIN
        PAYROLL_ENTRY_PKG.ADD_DEPARTMENT(10, 'IT', 100);
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    BEGIN
        PAYROLL_ENTRY_PKG.ADD_DEPARTMENT(20, 'Human Resources', 200);
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    BEGIN
        PAYROLL_ENTRY_PKG.ADD_DEPARTMENT(30, 'Executive', 100);
    EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL;
    END;

    -- D. Use Package to add Employees
    PAYROLL_ENTRY_PKG.ADD_EMPLOYEE(101, 'Steven', 'King', 'SKING', SYSDATE-3650,
'AD_PRES', 24000, 30, 1);
    PAYROLL_ENTRY_PKG.ADD_EMPLOYEE(102, 'Nina', 'Kochhar', 'NKOCHHAR', SYSDATE-2000,
'HR_REP', 17000, 20, 1);
    PAYROLL_ENTRY_PKG.ADD_EMPLOYEE(103, 'Lex', 'De Haan', 'LDEHAAN', SYSDATE-1500,
'IT_PROG', 9000, 10, 1);
    PAYROLL_ENTRY_PKG.ADD_EMPLOYEE(104, 'Alexander', 'Hunold', 'AHUNOLD', SYSDATE-100,
'IT_PROG', 6000, 10, 2);

    DBMS_OUTPUT.PUT_LINE('Data Entry Complete (Duplicates Ignored).');
END;
/
```



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSI
1	101	Steven	King	SKING	(null)	12/10/2015, 7:36:01 PM	AD_PRES	24000	
2	102	Nina	Kochhar	NKOCHHAR	(null)	6/16/2020, 7:36:01 PM	HR_REP	17000	
3	103	Lex	De Haan	LDEHAAN	(null)	10/29/2021, 7:36:01 PM	IT_PROG	9000	
4	104	Alexander	Hunold	AHUNOLD	(null)	8/29/2025, 7:36:01 PM	IT_PROG	6000	

Figure3: Employee table

5. Data Update via PL/SQL Package

5.1 Package specification

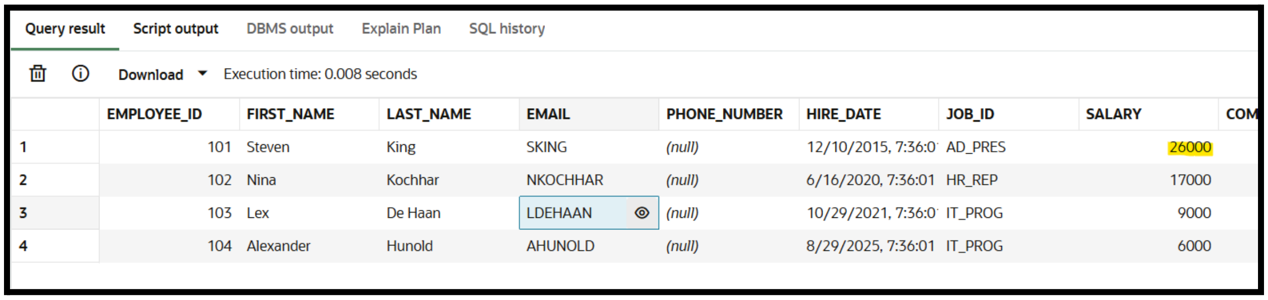
```
CREATE OR REPLACE PACKAGE PAYROLL_MANAGER_PKG AS
    -- Step 5: Procedure to Update Salary
    PROCEDURE UPDATE_SALARY(
        p_emp_id IN NUMBER,
        p_new_salary IN NUMBER
    );
END PAYROLL_MANAGER_PKG;
/
```

5.2 Package body

```
CREATE OR REPLACE PACKAGE BODY PAYROLL_MANAGER_PKG AS

    -- Implementation of Salary Update
    PROCEDURE UPDATE_SALARY(p_emp_id IN NUMBER, p_new_salary IN NUMBER) IS
        v_old_salary NUMBER;
    BEGIN
        -- First, let's check the current salary to print a nice message later
        SELECT SALARY INTO v_old_salary
        FROM EMPLOYEES
        WHERE EMPLOYEE_ID = p_emp_id;

        -- Perform the Update
        UPDATE EMPLOYEES
        SET SALARY = p_new_salary
        WHERE EMPLOYEE_ID = p_emp_id;

        -- Commit the change
        COMMIT;

        DBMS_OUTPUT.PUT_LINE('Salary for Employee ' || p_emp_id ||
                             ' updated from ' || v_old_salary ||
                             ' to ' || p_new_salary);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: Employee ID ' || p_emp_id || ' not found.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error updating salary: ' || SQLERRM);
    END UPDATE_SALARY;
END PAYROLL_MANAGER_PKG;
/
```

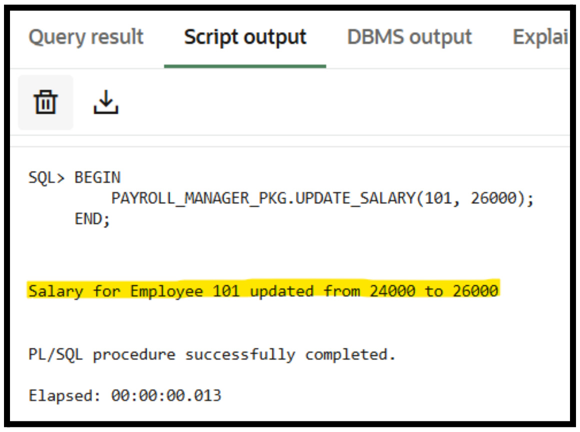
5.3 Execution code

```
SET SERVEROUTPUT ON;
BEGIN
    PAYROLL_MANAGER_PKG.UPDATE_SALARY(101, 26000);
END;
/
```



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COM
1	101	Steven	King	SKING	(null)	12/10/2015, 7:36:0	AD_PRES	26000	
2	102	Nina	Kochhar	NKOCHHAR	(null)	6/16/2020, 7:36:01	HR_REP	17000	
3	103	Lex	De Haan	LDEHAAN	(null)	10/29/2021, 7:36:0	IT_PROG	9000	
4	104	Alexander	Hunold	AHUNOLD	(null)	8/29/2025, 7:36:01	IT_PROG	6000	

Figure 4: Salary of employee 101 (24000 to 26000)



```
SQL> BEGIN
         PAYROLL_MANAGER_PKG.UPDATE_SALARY(101, 26000);
     END;


Salary for Employee 101 updated from 24000 to 26000


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013
```

Figure 5: Script output of execution code

6. Data Deletion

6.1 Updated Package Specification

```
CREATE OR REPLACE PACKAGE PAYROLL_MANAGER_PKG AS
    -- Step 5: Procedure to Update Salary
    PROCEDURE UPDATE_SALARY(
        p_emp_id IN NUMBER,
        p_new_salary IN NUMBER
    );

    -- Step 6: Procedure to Delete an Employee
    PROCEDURE REMOVE_EMPLOYEE(
        p_emp_id IN NUMBER
    );
END PAYROLL_MANAGER_PKG;
/
```

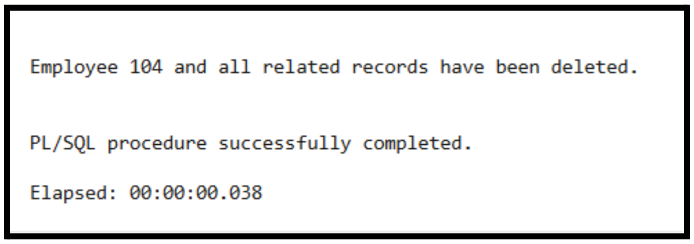
6.2 Updated Package Body

```
CREATE OR REPLACE PACKAGE BODY PAYROLL_MANAGER_PKG AS

    -- Update Implementation (From Step 5)
    PROCEDURE UPDATE_SALARY(p_emp_id IN NUMBER, p_new_salary IN NUMBER) IS
        v_old_salary NUMBER;
    BEGIN
        SELECT SALARY INTO v_old_salary FROM EMPLOYEES WHERE EMPLOYEE_ID = p_emp_id;

        UPDATE EMPLOYEES
        SET SALARY = p_new_salary
        WHERE EMPLOYEE_ID = p_emp_id;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Salary for Employee ' || p_emp_id || ' updated.');
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: Employee ID ' || p_emp_id || ' not found.');
    END UPDATE_SALARY;

    -- Delete Implementation (Step 6)
    PROCEDURE REMOVE_EMPLOYEE(p_emp_id IN NUMBER) IS
        v_count NUMBER;
    BEGIN
```

```
        -- Check if employee exists first
        SELECT COUNT(*) INTO v_count FROM EMPLOYEES WHERE EMPLOYEE_ID = p_emp_id;

        IF v_count = 0 THEN
            DBMS_OUTPUT.PUT_LINE('Employee ' || p_emp_id || ' does not exist.');
        ELSE
            -- 1. Clean up child tables to prevent Foreign Key errors
            DELETE FROM JOB_HISTORY WHERE EMPLOYEE_ID = p_emp_id;
            DELETE FROM LEAVE_REQUESTS WHERE EMPLOYEE_ID = p_emp_id;
            DELETE FROM ATTENDANCE_LOGS WHERE EMPLOYEE_ID = p_emp_id;
            DELETE FROM EMPLOYEE_DEDUCTIONS WHERE EMPLOYEE_ID = p_emp_id;
            DELETE FROM PAYROLL_TRANSACTIONS WHERE EMPLOYEE_ID = p_emp_id;

            -- 2. Delete the actual Employee
            DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = p_emp_id;

            COMMIT;
            DBMS_OUTPUT.PUT_LINE('Employee ' || p_emp_id || ' and all related records
have been deleted.');
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK; -- Undo changes if something goes wrong
            DBMS_OUTPUT.PUT_LINE('Error deleting employee: ' || SQLERRM);
    END REMOVE_EMPLOYEE;

END PAYROLL_MANAGER_PKG;
/
```

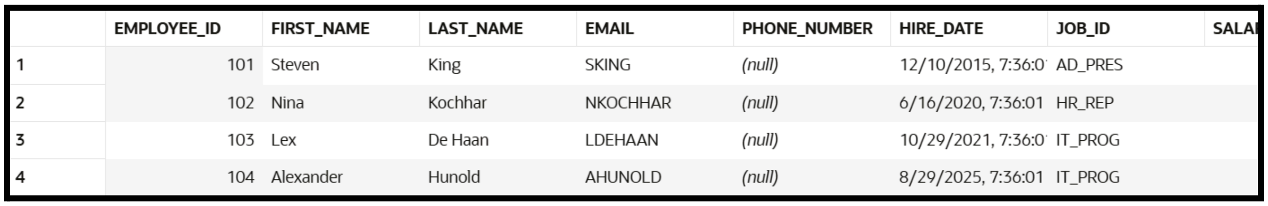
6.3 Execution Code

```
SET SERVEROUTPUT ON;
BEGIN
    PAYROLL_MANAGER_PKG.REMOVE_EMPLOYEE(104);
END;
/
```



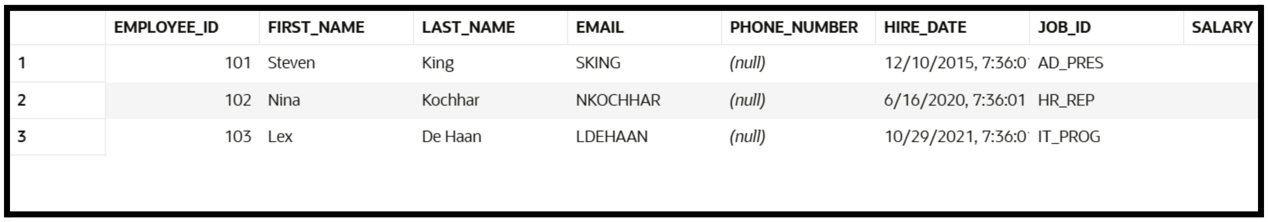
```
Employee 104 and all related records have been deleted.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.038
```

Figure 6: Script output of execution code (deletion)



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
1	101	Steven	King	SKING	(null)	12/10/2015, 7:36:0	AD_PRES	
2	102	Nina	Kochhar	NKOCHHAR	(null)	6/16/2020, 7:36:01	HR_REP	
3	103	Lex	De Haan	LDEHAAN	(null)	10/29/2021, 7:36:0	IT_PROG	
4	104	Alexander	Hunold	AHUNOLD	(null)	8/29/2025, 7:36:01	IT_PROG	

Figure 7: Employee table before deletion



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	101	Steven	King	SKING	(null)	12/10/2015, 7:36:0	AD_PRES	
2	102	Nina	Kochhar	NKOCHHAR	(null)	6/16/2020, 7:36:01	HR_REP	
3	103	Lex	De Haan	LDEHAAN	(null)	10/29/2021, 7:36:0	IT_PROG	

Figure 8: Employee table after deletion of emp 104

7. Automating Data Entry with a Trigger

7.1 Create a Sequence (Helper Object)

```
-- Create a sequence starting at 100
CREATE SEQUENCE JOB_HISTORY_SEQ
START WITH 100
INCREMENT BY 1;
```

7.2 Create the Trigger

```
CREATE OR REPLACE TRIGGER TRG_LOG_NEW_HIRE
AFTER INSERT ON EMPLOYEES
FOR EACH ROW
BEGIN
    -- Insert a record into the transaction table automatically
    INSERT INTO JOB_HISTORY (
        HISTORY_ID,
        EMPLOYEE_ID,
        START_DATE,
        END_DATE,
        JOB_ID,
        DEPARTMENT_ID
    ) VALUES (
        JOB_HISTORY_SEQ.NEXTVAL, -- Generate new ID
        :NEW.EMPLOYEE_ID,        -- Use the ID of the new employee
        :NEW.HIRE_DATE,          -- Use the hire date
        NULL,                    -- No end date yet
        :NEW.JOB_ID,             -- Use the job they were hired for
        :NEW.DEPARTMENT_ID       -- Use the department they joined
    );
END;
/
```

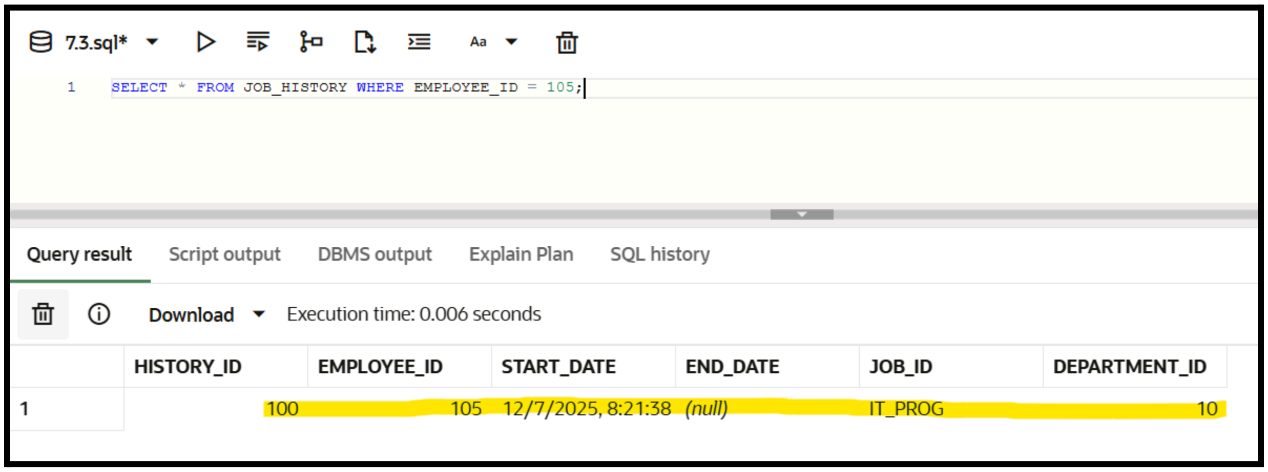
7.3 Test the Trigger

Insert a New Employee (Definition Table)

```
BEGIN
    -- We use our package from Step 4, or a standard insert
    PAYROLL_ENTRY_PKG.ADD_EMPLOYEE(
        105, 'Bruce', 'Ernst', 'BERNST', SYSDATE, 'IT_PROG', 6000, 10, 1
    );
END;
/
```

Verify the Automatic Transaction

```
SELECT * FROM JOB_HISTORY WHERE EMPLOYEE_ID = 105;
```



```
SELECT * FROM JOB_HISTORY WHERE EMPLOYEE_ID = 105;
```

Execution time: 0.006 seconds

	HISTORY_ID	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	100	105	12/7/2025, 8:21:38	*(null)*	IT_PROG	10

Figure 9: JOB_HISTORY table

8. Dynamic Reporting with PL/SQL

8.1 Create the Report Package Specification

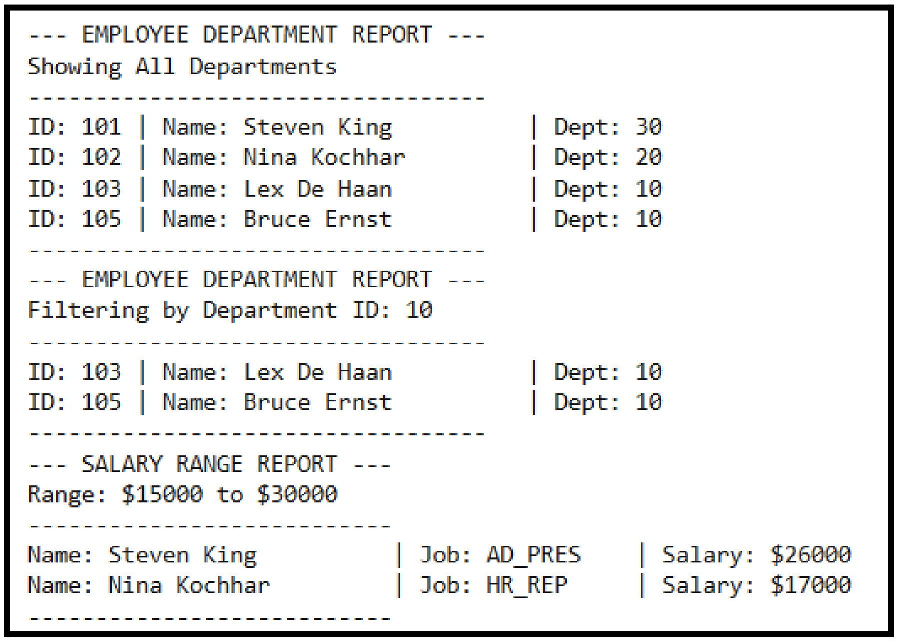
```
CREATE OR REPLACE PACKAGE PAYROLL_REPORT_PKG AS

    -- Report 1: Filter Employees by Department (Dynamic)
    PROCEDURE EMP_BY_DEPT_REPORT(
        p_dept_id IN NUMBER DEFAULT NULL -- Default NULL means "Show All"
    );

    -- Report 2: Filter Employees by Salary Range (Dynamic)
    PROCEDURE SALARY_RANGE_REPORT(
        p_min_sal IN NUMBER,
        p_max_sal IN NUMBER
    );

END PAYROLL_REPORT_PKG;
/
```

8.2 Create the Report Package Body

```
CREATE OR REPLACE PACKAGE BODY PAYROLL_REPORT_PKG AS

    -- Implementation of Report 1
    PROCEDURE EMP_BY_DEPT_REPORT(p_dept_id IN NUMBER DEFAULT NULL) IS
        -- Cursor that adapts to the input
        CURSOR c_emps IS
            SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT_ID
            FROM EMPLOYEES
            WHERE (p_dept_id IS NULL OR DEPARTMENT_ID = p_dept_id)
            ORDER BY EMPLOYEE_ID;

        v_rec c_emps%ROWTYPE;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('--- EMPLOYEE DEPARTMENT REPORT ---');
        IF p_dept_id IS NOT NULL THEN
             DBMS_OUTPUT.PUT_LINE('Filtering by Department ID: ' || p_dept_id);
        ELSE
             DBMS_OUTPUT.PUT_LINE('Showing All Departments');
        END IF;
        DBMS_OUTPUT.PUT_LINE('----------------------------------');

        OPEN c_emps;
        LOOP
            FETCH c_emps INTO v_rec;
            EXIT WHEN c_emps%NOTFOUND;

            DBMS_OUTPUT.PUT_LINE(
                'ID: ' || v_rec.EMPLOYEE_ID ||
                ' | Name: ' || RPAD(v_rec.FIRST_NAME || ' ' || v_rec.LAST_NAME, 20) ||
                ' | Dept: ' || v_rec.DEPARTMENT_ID
            );
        END LOOP;
        CLOSE c_emps;
        DBMS_OUTPUT.PUT_LINE('----------------------------------');
    END EMP_BY_DEPT_REPORT;

    -- Implementation of Report 2
    PROCEDURE SALARY_RANGE_REPORT(p_min_sal IN NUMBER, p_max_sal IN NUMBER) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('--- SALARY RANGE REPORT ---');
        DBMS_OUTPUT.PUT_LINE('Range: $' || p_min_sal || ' to $' || p_max_sal);
        DBMS_OUTPUT.PUT_LINE('---------------------------');

        FOR r IN (
            SELECT FIRST_NAME, LAST_NAME, SALARY, JOB_ID
            FROM EMPLOYEES
            WHERE SALARY BETWEEN p_min_sal AND p_max_sal
            ORDER BY SALARY DESC
        ) LOOP
            DBMS_OUTPUT.PUT_LINE(
                'Name: ' || RPAD(r.FIRST_NAME || ' ' || r.LAST_NAME, 20) ||
                ' | Job: ' || RPAD(r.JOB_ID, 10) ||
                ' | Salary: $' || r.SALARY
            );
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('---------------------------');
    END SALARY_RANGE_REPORT;

END PAYROLL_REPORT_PKG;
/
```

8.3 Execution (Running the Dynamic Reports)

```
SET SERVEROUTPUT ON;

BEGIN
    -- Test A: Get all employees (Sending NULL constraint)
    PAYROLL_REPORT_PKG.EMP_BY_DEPT_REPORT(NULL);

    -- Test B: Get only IT employees (Sending '10' as constraint)
    PAYROLL_REPORT_PKG.EMP_BY_DEPT_REPORT(10);

    -- Test C: Get High Earners (Salary between 15,000 and 30,000)
    PAYROLL_REPORT_PKG.SALARY_RANGE_REPORT(15000, 30000);
END;
/
```



```
--- EMPLOYEE DEPARTMENT REPORT ---
Showing All Departments
----------------------------------
ID: 101 | Name: Steven King          | Dept: 30
ID: 102 | Name: Nina Kochhar         | Dept: 20
ID: 103 | Name: Lex De Haan          | Dept: 10
ID: 105 | Name: Bruce Ernst          | Dept: 10
----------------------------------
--- EMPLOYEE DEPARTMENT REPORT ---
Filtering by Department ID: 10
----------------------------------
ID: 103 | Name: Lex De Haan          | Dept: 10
ID: 105 | Name: Bruce Ernst          | Dept: 10
----------------------------------
--- SALARY RANGE REPORT ---
Range: $15000 to $30000
---------------------------
Name: Steven King         | Job: AD_PRES     | Salary: $26000
Name: Nina Kochhar        | Job: HR_REP      | Salary: $17000
---------------------------
```

Figure 10: Script output about dynamic report

9. Proof of 3rd Normal Form

Relational Schema

1. **REGIONS** (**REGION_ID**, REGION_NAME)
2. **COUNTRIES** (**COUNTRY_ID**, COUNTRY_NAME, *REGION_ID*)
3. **LOCATIONS** (**LOCATION_ID**, STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, *COUNTRY_ID*)
4. **DEPARTMENTS** (**DEPARTMENT_ID**, DEPARTMENT_NAME, *LOCATION_ID*)
5. **JOBS** (**JOB_ID**, JOB_TITLE, MIN_SALARY, MAX_SALARY)
6. **STATUSES** (**STATUS_ID**, STATUS_DESC)
7. **EMPLOYEES** (**EMPLOYEE_ID**, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, SALARY, COMMISSION_PCT, *JOB_ID*, *MANAGER_ID*, *DEPARTMENT_ID*, *STATUS_ID*)
8. **JOB_HISTORY** (**HISTORY_ID**, *EMPLOYEE_ID*, START_DATE, END_DATE, *JOB_ID*, *DEPARTMENT_ID*)
9. **LEAVE_TYPES** (**LEAVE_TYPE_ID**, LEAVE_NAME)
10. **LEAVE_REQUESTS** (**REQUEST_ID**, *EMPLOYEE_ID*, *LEAVE_TYPE_ID*, START_DATE, END_DATE, REASON, APPROVAL_STATUS)
11. **ATTENDANCE_LOGS** (**LOG_ID**, *EMPLOYEE_ID*, LOG_DATE, CHECK_IN_TIME, CHECK_OUT_TIME)
12. **DEDUCTION_TYPES** (**DEDUCTION_ID**, DEDUCTION_NAME, DESCRIPTION)
13. **EMPLOYEE_DEDUCTIONS** (**EMP_DEDUCTION_ID**, *EMPLOYEE_ID*, *DEDUCTION_ID*, DEDUCTION_AMOUNT, DEDUCTION_PERCENT)
14. **PAYROLL_TRANSACTIONS** (**TRANSACTION_ID**, *EMPLOYEE_ID*, TRANSACTION_DATE, PERIOD_START, PERIOD_END, GROSS_PAY, TOTAL_DEDUCTIONS, NET_PAY)

Our Payroll Management System satisfies 3NF by separating data into distinct entities (**EMPLOYEES, DEPARTMENTS, LOCATIONS, JOBS**) to prevent data redundancy.

Example 1: Removal of Location Dependency

The Violation (Not 3NF): If we stored **City** and **State** inside the **EMPLOYEES** table, we would have a transitive dependency:

- Employee_ID (PK) → Department_ID → City.
- This creates redundancy. If the IT department moves to a new city, we would have to update every single IT employee's record.

The Solution (Our 3NF Design):

- We created a **LOCATIONS** table.
- We created a **DEPARTMENTS** table that links to LOCATION_ID.

- The **EMPLOYEES** table links only to DEPARTMENT_ID.
- Result: The City attribute is stored in only one place. EMPLOYEES depends on DEPARTMENTS, and DEPARTMENTS depends on LOCATIONS. The transitive link is broken.

Example 2: Removal of Job Detail Dependency

The Violation (Not 3NF):
If we stored **Job_Title**, **Min_Salary**, and **Max_Salary** directly in the **EMPLOYEES** table.

- Employee_ID (PK) → Job_Title → Min_Salary.

The Solution (Our 3NF Design):

- We created a separate **JOBS** table (Primary Key: JOB_ID).
- The **EMPLOYEES** table references **JOB_ID** via a **Foreign Key.**
- Result: If the salary range for "Programmer" changes, we update it once in the JOBS table, not in every employee record.

Another proof of our database in 3rd normal form via visual aid E-R diagram

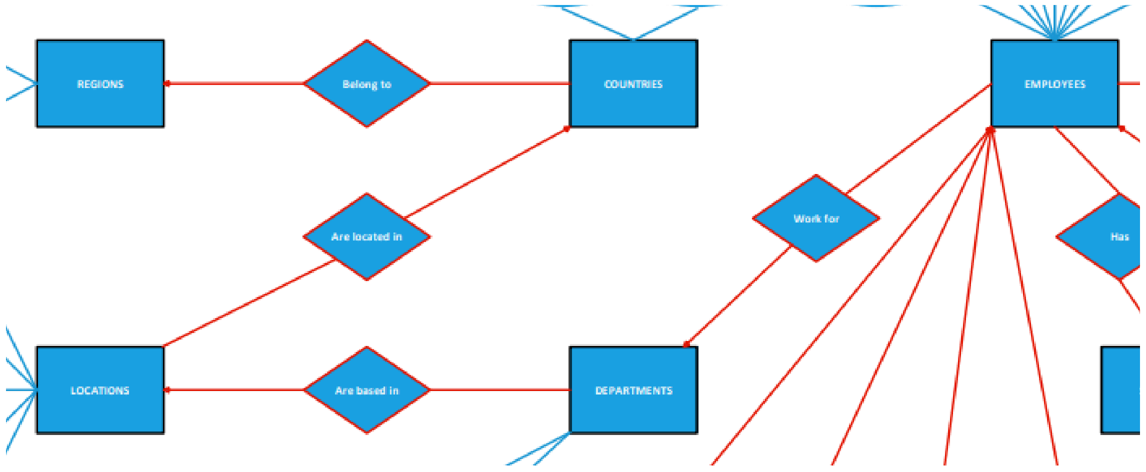


Figure 11: A section of the E-R diagram

When we look at the E-R diagram (figure 11), our **arrows look like a chain**, **not a spiderweb**. This proves it is in a 3RD normal form.

10. Delete Duplicate Records

10.1 Create the Cleanup Package Specification

```
CREATE OR REPLACE PACKAGE DATA_CLEANUP_PKG AS
    -- Procedure to remove duplicate employees based on Email
    PROCEDURE REMOVE_DUPLICATE_EMPLOYEES;
END DATA_CLEANUP_PKG;
/
```

10.2 Create the Cleanup Package Body

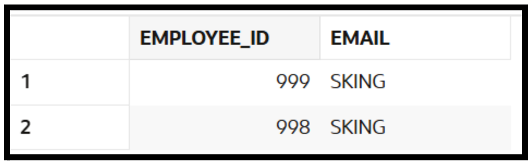
```
CREATE OR REPLACE PACKAGE BODY DATA_CLEANUP_PKG AS

    PROCEDURE REMOVE_DUPLICATE_EMPLOYEES IS
    BEGIN
        -- Loop through all duplicate employees (Same Email, but not the oldest one)
        FOR r IN (
            SELECT EMPLOYEE_ID, EMAIL
            FROM EMPLOYEES A
            WHERE ROWID > (
                SELECT MIN(ROWID)
                FROM EMPLOYEES B
                WHERE A.EMAIL = B.EMAIL
            )
        ) LOOP
            -- 1. First, delete the automated history record (created by Step 7
trigger)
            DELETE FROM JOB_HISTORY WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;

            -- 2. Also delete any other potential child records (just in case)
            DELETE FROM LEAVE_REQUESTS WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;
            DELETE FROM ATTENDANCE_LOGS WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;
            DELETE FROM PAYROLL_TRANSACTIONS WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;
            DELETE FROM EMPLOYEE_DEDUCTIONS WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;

            -- 3. Now we can safely delete the duplicate Employee
            DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = r.EMPLOYEE_ID;

            DBMS_OUTPUT.PUT_LINE('Deleted duplicate Employee ID: ' || r.EMPLOYEE_ID ||
' (Email: ' || r.EMAIL || ')');
        END LOOP;

        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Cleanup procedure finished.');

    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END REMOVE_DUPLICATE_EMPLOYEES;

END DATA_CLEANUP_PKG;
/
```

10.3 Test Scenario (Creating "Bad" Data)

```
SET SERVEROUTPUT ON;

BEGIN
    -- 1. Insert First Logical Duplicate (ID 999)
    INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE,
JOB_ID, DEPARTMENT_ID, STATUS_ID)
    VALUES (999, 'Steven', 'King', 'SKING', SYSDATE, 'AD_PRES', 30, 1);

    -- 2. Insert Second Logical Duplicate (ID 998)
    -- We repeat the INSERT statement
    INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE,
JOB_ID, DEPARTMENT_ID, STATUS_ID)
    VALUES (998, 'Steven', 'King', 'SKING', SYSDATE, 'AD_PRES', 30, 1);

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Two duplicate records created for testing.');
END;
/
```

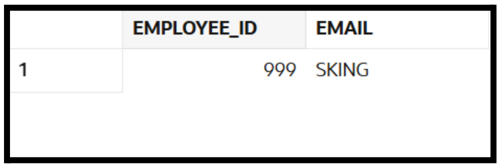
10.4 Execute the Cleanup

```
SET SERVEROUTPUT ON;
BEGIN
    DATA_CLEANUP_PKG.REMOVE_DUPLICATE_EMPLOYEES;
END;
/
```



	EMPLOYEE_ID	EMAIL
1	999	SKING
2	998	SKING

Figure 11: Before the execution cleanup

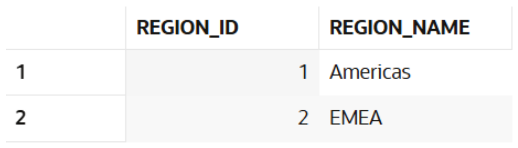


	EMPLOYEE_ID	EMAIL
1	999	SKING

Figure 12: After the execution cleanup

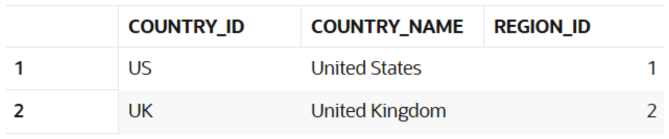
Additional: Tables

1- Region



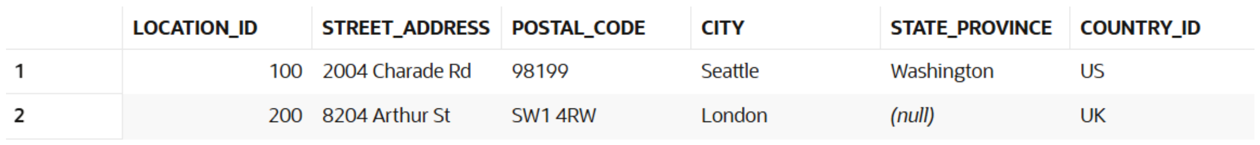
	REGION_ID	REGION_NAME
1	1	Americas
2	2	EMEA

2- Countries



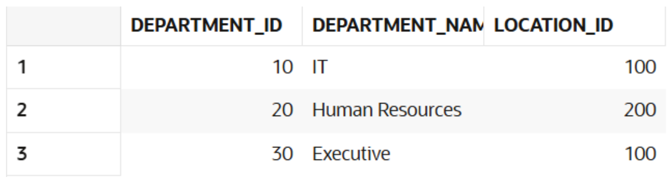
	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	US	United States	1
2	UK	United Kingdom	2

3- Locations



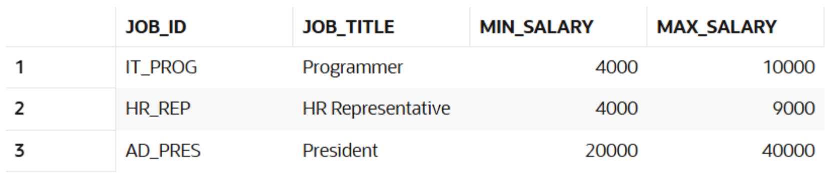
	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	100	2004 Charade Rd	98199	Seattle	Washington	US
2	200	8204 Arthur St	SW1 4RW	London	*(null)*	UK

4- Departments



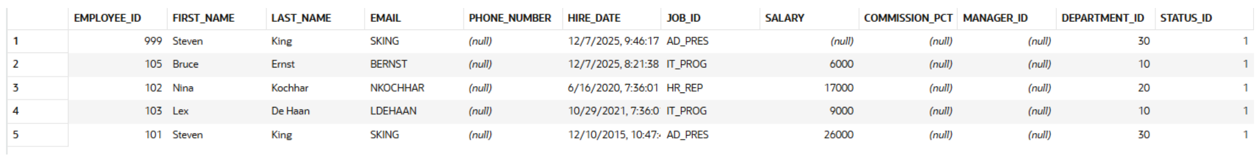
	DEPARTMENT_ID	DEPARTMENT_NAM	LOCATION_ID
1	10	IT	100
2	20	Human Resources	200
3	30	Executive	100

5- Jobs



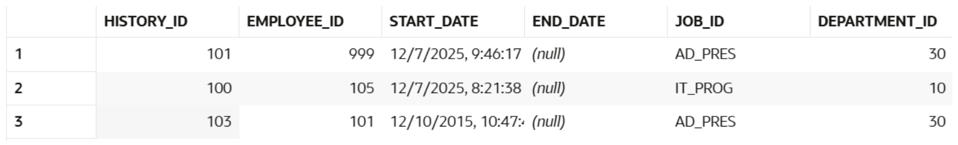
	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	IT_PROG	Programmer	4000	10000
2	HR_REP	HR Representative	4000	9000
3	AD_PRES	President	20000	40000

6- Employees



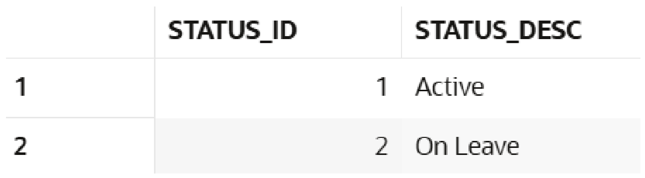
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	STATUS_ID
1	999	Steven	King	SKING	*(null)*	12/7/2025, 9:46:17	AD_PRES	*(null)*	*(null)*	*(null)*	30	1
2	105	Bruce	Ernst	BERNST	*(null)*	12/7/2025, 8:21:38	IT_PROG	6000	*(null)*	*(null)*	10	1
3	102	Nina	Kochhar	NKOCHHAR	*(null)*	6/16/2020, 7:36:01	HR_REP	17000	*(null)*	*(null)*	20	1
4	103	Lex	De Haan	LDEHAAN	*(null)*	10/29/2021, 7:36:0	IT_PROG	9000	*(null)*	*(null)*	10	1
5	101	Steven	King	SKING	*(null)*	12/10/2015, 10:47:	AD_PRES	26000	*(null)*	*(null)*	30	1

7- Job_history



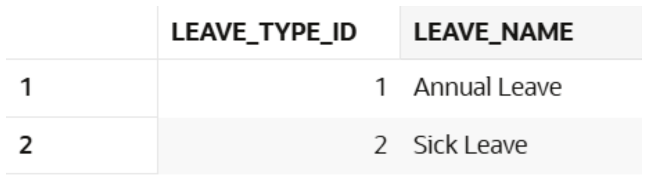
	HISTORY_ID	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	101	999	12/7/2025, 9:46:17	*(null)*	AD_PRES	30
2	100	105	12/7/2025, 8:21:38	*(null)*	IT_PROG	10
3	103	101	12/10/2015, 10:47:	*(null)*	AD_PRES	30

8- Statuses



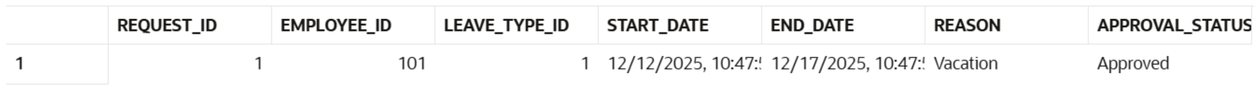
	STATUS_ID	STATUS_DESC
1	1	Active
2	2	On Leave

9- Leave_types



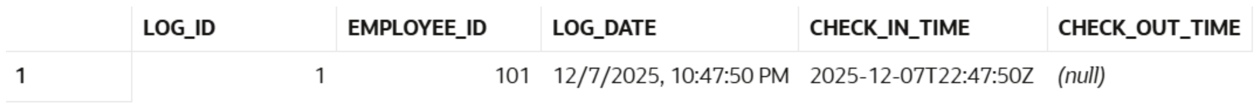
	LEAVE_TYPE_ID	LEAVE_NAME
1	1	Annual Leave
2	2	Sick Leave

10- Leave_requests



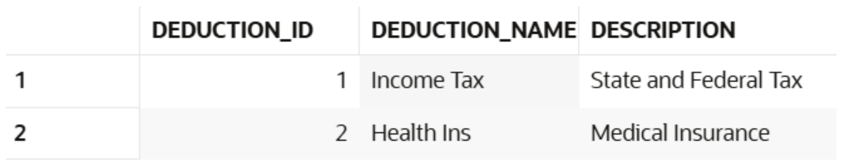
	REQUEST_ID	EMPLOYEE_ID	LEAVE_TYPE_ID	START_DATE	END_DATE	REASON	APPROVAL_STATUS
1	1	101	1	12/12/2025, 10:47:	12/17/2025, 10:47:	Vacation	Approved

11- Attendance_logs



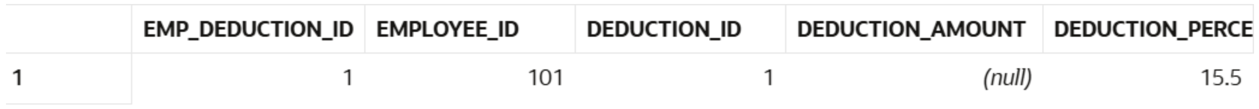
	LOG_ID	EMPLOYEE_ID	LOG_DATE	CHECK_IN_TIME	CHECK_OUT_TIME
1	1	101	12/7/2025, 10:47:50 PM	2025-12-07T22:47:50Z	*(null)*

12- Deduction_types



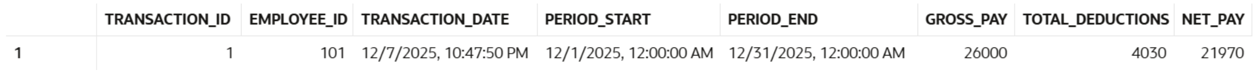
	DEDUCTION_ID	DEDUCTION_NAME	DESCRIPTION
1	1	Income Tax	State and Federal Tax
2	2	Health Ins	Medical Insurance

13- Employee_deduction



	EMP_DEDUCTION_ID	EMPLOYEE_ID	DEDUCTION_ID	DEDUCTION_AMOUNT	DEDUCTION_PERCE
1	1	101	1	*(null)*	15.5

14- Payroll_transaction



	TRANSACTION_ID	EMPLOYEE_ID	TRANSACTION_DATE	PERIOD_START	PERIOD_END	GROSS_PAY	TOTAL_DEDUCTIONS	NET_PAY
1	1	101	12/7/2025, 10:47:50 PM	12/1/2025, 12:00:00 AM	12/31/2025, 12:00:00 AM	26000	4030	21970