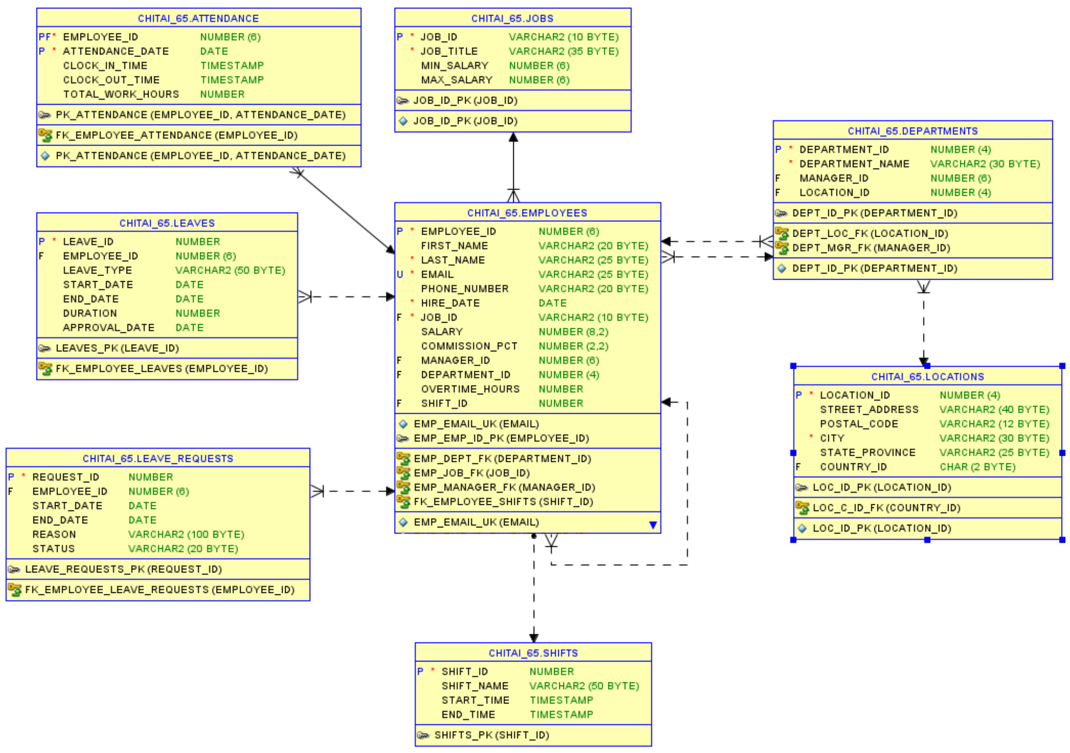
Employee Attendance Tracking

Chita Ionut-Cristian

This employee attendance tracking database was created in order to make managing a large array of employees easier, it can track many important managerial variables like attendance, leaves and leave requests, and even worker shifts if needed. In order to achieve all of that, the original schema from both the first and second semester had to be modified by adding a few more tables, here is the modified database schema (Note that it there are more tables, adding more functionality to the database itself, but I chose to show only the ones relevant to my project).


CHITAI_65.ATTENDANCE
PF* EMPLOYEE_ID NUMBER (6)
P * ATTENDANCE_DATE DATE
CLOCK_IN_TIME TIMESTAMP
CLOCK_OUT_TIME TIMESTAMP
TOTAL_WORK_HOURS NUMBER
PK_ATTENDANCE (EMPLOYEE_ID, ATTENDANCE_DATE)
FK_EMPLOYEE_ATTENDANCE (EMPLOYEE_ID)
PK_ATTENDANCE (EMPLOYEE_ID, ATTENDANCE_DATE)
CHITAI_65.JOBS
P * JOB_ID VARCHAR2 (10 BYTE)
* JOB_TITLE VARCHAR2 (35 BYTE)
MIN_SALARY NUMBER (6)
MAX_SALARY NUMBER (6)
JOB_ID_PK (JOB_ID)
JOB_ID_PK (JOB_ID)
CHITAI_65.DEPARTMENTS
P * DEPARTMENT_ID NUMBER (4)
* DEPARTMENT_NAME VARCHAR2 (30 BYTE)
F MANAGER_ID NUMBER (6)
F LOCATION_ID NUMBER (4)
DEPT_ID_PK (DEPARTMENT_ID)
DEPT_LOC_FK (LOCATION_ID)
DEPT_MGR_FK (MANAGER_ID)
DEPT_ID_PK (DEPARTMENT_ID)
CHITAI_65.LEAVES
P * LEAVE_ID NUMBER
F EMPLOYEE_ID NUMBER (6)
LEAVE_TYPE VARCHAR2 (50 BYTE)
START_DATE DATE
END_DATE DATE
DURATION NUMBER
APPROVAL_DATE DATE
LEAVES_PK (LEAVE_ID)
FK_EMPLOYEE_LEAVES (EMPLOYEE_ID)
CHITAI_65.EMPLOYEES
P * EMPLOYEE_ID NUMBER (6)
FIRST_NAME VARCHAR2 (20 BYTE)
* LAST_NAME VARCHAR2 (25 BYTE)
U * EMAIL VARCHAR2 (25 BYTE)
PHONE_NUMBER VARCHAR2 (20 BYTE)
* HIRE_DATE DATE
F * JOB_ID VARCHAR2 (10 BYTE)
SALARY NUMBER (8,2)
COMMISSION_PCT NUMBER (2,2)
F MANAGER_ID NUMBER (6)
F DEPARTMENT_ID NUMBER (4)
OVERTIME_HOURS NUMBER
F SHIFT_ID NUMBER
EMP_EMAIL_UK (EMAIL)
EMP_EMP_ID_PK (EMPLOYEE_ID)
EMP_DEPT_FK (DEPARTMENT_ID)
EMP_JOB_FK (JOB_ID)
EMP_MANAGER_FK (MANAGER_ID)
FK_EMPLOYEE_SHIFTS (SHIFT_ID)
EMP_EMAIL_UK (EMAIL)
CHITAI_65.LOCATIONS
P * LOCATION_ID NUMBER (4)
STREET_ADDRESS VARCHAR2 (40 BYTE)
POSTAL_CODE VARCHAR2 (12 BYTE)
* CITY VARCHAR2 (30 BYTE)
STATE_PROVINCE VARCHAR2 (25 BYTE)
F COUNTRY_ID CHAR (2 BYTE)
LOC_ID_PK (LOCATION_ID)
LOC_C_ID_FK (COUNTRY_ID)
LOC_ID_PK (LOCATION_ID)
CHITAI_65.LEAVE_REQUESTS
P * REQUEST_ID NUMBER
F EMPLOYEE_ID NUMBER (6)
START_DATE DATE
END_DATE DATE
REASON VARCHAR2 (100 BYTE)
STATUS VARCHAR2 (20 BYTE)
LEAVE_REQUESTS_PK (REQUEST_ID)
FK_EMPLOYEE_LEAVE_REQUESTS (EMPLOYEE_ID)
CHITAI_65.SHIFTS
P * SHIFT_ID NUMBER
SHIFT_NAME VARCHAR2 (50 BYTE)
START_TIME TIMESTAMP
END_TIME TIMESTAMP
SHIFTS_PK (SHIFT_ID)

The **"ATTENDANCE"** table is responsible for recording and managing employee attendance information. It tracks clock-in and clock-out times, allowing for the calculation of total work hours. The **"LEAVE_REQUESTS"** table is used to manage employee leave requests and track approved leaves within the organization, allowing for efficient leave management. The **"LEAVES"** table is utilized to track and manage employee leaves and absences within the organization. It serves as a record of all types of employee leaves, including vacation, sick leave, or other authorized time off. The **"SHIFTS"** table is designed to manage and organize different work shifts within the organization. It allows for efficient scheduling and assignment of employees to specific shifts.

These tables play a crucial role in attendance management systems, providing valuable data for reporting, analysis, and decision-making processes within the organization. They contribute to efficient employee scheduling, optimized resource utilization, comprehensive leave management, and facilitate smooth operational coverage throughout different shifts and time periods.

A. Using execute immediate

Create a new table dynamically and insert data into it. Retrieve total salary using a function at the row level

```
DECLARE
  v_table_name VARCHAR2(100) := 'NEW_TABLE';
  v_count NUMBER;
  v_total_salary NUMBER;
BEGIN

  EXECUTE IMMEDIATE 'CREATE TABLE ' || v_table_name || ' (id NUMBER, name VARCHAR2(50))';

  EXECUTE IMMEDIATE 'INSERT INTO ' || v_table_name || ' VALUES (1, ''John'')';
  EXECUTE IMMEDIATE 'INSERT INTO ' || v_table_name || ' VALUES (2, ''Jane'')';

  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || v_table_name INTO v_count;
  DBMS_OUTPUT.PUT_LINE('Number of rows in ' || v_table_name || ': ' || v_count);

  SELECT SUM(salary)
    INTO v_total_salary
    FROM employees;

  DBMS_OUTPUT.PUT_LINE('Total salary: ' || v_total_salary);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

B. Alternative and repetitive structures (IF, CASE, FOR, LOOP, WHILE)

IF Statement: Update the attendance status of an employee based on their clock-in time. If the clock-in time is before 9:00 AM, the status should be marked as ”On Time,” otherwise, it should be marked as ”Late.”

```
DECLARE
  v_employee_id NUMBER := 1001;
  v_clock_in_time DATE := TO_DATE('2023-05-15 08:55:00', 'YYYY-MM-DD HH24:MI:SS');
  v_attendance_status VARCHAR2(20);
BEGIN
  IF v_clock_in_time < TO_DATE('2023-05-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS') THEN
    v_attendance_status := 'On Time';
  ELSE
    v_attendance_status := 'Late';
  END IF;

  UPDATE attendance
  SET status = v_attendance_status
  WHERE employee_id = v_employee_id;

  DBMS_OUTPUT.PUT_LINE('Attendance status updated successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

CASE Statement: Retrieve the leave type for an employee based on the number of days they have taken leave. If the number of leave days is less than or equal to 5, the leave type should be marked as ”Sick Leave.” Otherwise, it should be marked as ”Vacation.”

```
DECLARE
  v_employee_id NUMBER := 1002;
  v_leave_days NUMBER := 3;
  v_leave_type VARCHAR2(20);
BEGIN
  CASE
    WHEN v_leave_days <= 5 THEN
      v_leave_type := 'Sick Leave';
    ELSE
      v_leave_type := 'Vacation';
  END CASE;

  SELECT leave_type
  INTO v_leave_type
  FROM leaves
  WHERE employee_id = v_employee_id;

  DBMS_OUTPUT.PUT_LINE('Leave type for employee ' || v_employee_id || ': ' || v_leave_type);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No leave information found for employee ' || v_employee_id);
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

FOR Statement with implicit cursor: Calculate the total number of employees in each department and display the result.

```
DECLARE
  v_department_id NUMBER;
  v_department_name departments.department_name%TYPE;
  v_employee_count NUMBER;
BEGIN
  FOR dept IN (SELECT department_id, department_name FROM departments) LOOP
    v_department_id := dept.department_id;
    v_department_name := dept.department_name;

    SELECT COUNT(*) INTO v_employee_count
    FROM employees
    WHERE department_id = v_department_id;

    DBMS_OUTPUT.PUT_LINE
('Department: ' || v_department_name || ', Employee Count: ' || v_employee_count);
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

LOOP Statement with explicit cursor: Update the attendance status for all employees whose clock-in time is missing in the attendance table. Set their status as "No Clock-In."

```
DECLARE
  v_employee_id employees.employee_id%TYPE;

  CURSOR c_missing_clock_in IS
    SELECT employee_id
    FROM employees
    WHERE employee_id NOT IN (SELECT employee_id FROM attendance);
```

```
BEGIN
  OPEN c_missing_clock_in;

  LOOP
    FETCH c_missing_clock_in INTO v_employee_id;

    EXIT WHEN c_missing_clock_in%NOTFOUND;

    UPDATE attendance
    SET status = 'No Clock-In'
    WHERE employee_id = v_employee_id;
  END LOOP;

  CLOSE c_missing_clock_in;

  DBMS_OUTPUT.PUT_LINE('Attendance status updated for employees with missing clock-in time.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

WHILE and FUNCTION: Calculate the total salary for all employees in a specific department using a WHILE loop and a custom function. The function should take an employee ID as a parameter and return the employee's salary. The WHILE loop should iterate through each employee in the department and accumulate their salaries to calculate the total salary.

```
DECLARE
  v_department_id NUMBER := 100; -- Specify the department ID for calculation
  v_employee_id employees.employee_id%TYPE;
  v_total_salary NUMBER := 0;

  FUNCTION get_employee_salary(p_employee_id IN employees.employee_id%TYPE)
    RETURN employees.salary%TYPE
  IS
    v_salary employees.salary%TYPE;
  BEGIN
    SELECT salary INTO v_salary
    FROM employees
    WHERE employee_id = p_employee_id;

    RETURN v_salary;
  END;
BEGIN
  FOR emp IN (SELECT employee_id FROM employees WHERE department_id = v_department_id) LOOP
    v_employee_id := emp.employee_id;
    v_total_salary := v_total_salary + get_employee_salary(v_employee_id);
  END LOOP;

  DBMS_OUTPUT.PUT_LINE
('Total salary for department ' || v_department_id || ': ' || v_total_salary);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

C. Data collections (index by table, nested table, varray)

Index-By Table: Store a list of employees who have taken leaves for each department.

```
DECLARE
  TYPE leave_list_type IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
  v_leave_list leave_list_type;
BEGIN
  FOR dept IN (SELECT department_id, department_name FROM departments) LOOP
    v_leave_list.DELETE; -- Clear the leave list for each department

    SELECT e.first_name || ' ' || e.last_name
    BULK COLLECT INTO v_leave_list
    FROM employees e
    JOIN leaves l ON e.employee_id = l.employee_id
    WHERE e.department_id = dept.department_id;

    DBMS_OUTPUT.PUT_LINE('Department: ' || dept.department_name);
    FOR i IN 1..v_leave_list.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('Employee: ' || v_leave_list(i));
    END LOOP;
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

Nested Table: Track the attendance records for employees who have irregular working hours. This requires storing multiple clock-in and clock-out times for each employee in a flexible manner.

```
DECLARE
  TYPE attendance_records IS TABLE OF attendance.clock_in_time%TYPE;
  v_employee_id employees.employee_id%TYPE := 1001;
  v_attendance attendance_records := attendance_records();
BEGIN
  SELECT clock_in_time
  BULK COLLECT INTO v_attendance
  FROM attendance
  WHERE employee_id = v_employee_id;

  FOR i IN 1..v_attendance.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Attendance Record ' || i || ': ' || v_attendance(i));
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

Varray: Track and store the skill ratings for each employee based on their attendance at work. The skill rating should be calculated based on the number of hours worked by the employee, with a range from 1 to 5.

```
DECLARE
  TYPE skill_ratings IS VARRAY(5) OF NUMBER;
  v_employee_id employees.employee_id%TYPE := 1001;
  v_hours_worked NUMBER;
  v_skill_rating NUMBER;
  v_ratings skill_ratings := skill_ratings();
BEGIN

  SELECT SUM(hours_worked)
  INTO v_hours_worked
  FROM attendance
  WHERE employee_id = v_employee_id;

  IF v_hours_worked <= 100 THEN
    v_skill_rating := 1;
  ELSIF v_hours_worked <= 200 THEN
    v_skill_rating := 2;
  ELSIF v_hours_worked <= 300 THEN
    v_skill_rating := 3;
  ELSIF v_hours_worked <= 400 THEN
    v_skill_rating := 4;
  ELSE
    v_skill_rating := 5;
  END IF;

  v_ratings(1) := v_skill_rating;
  DBMS_OUTPUT.PUT_LINE('Skill Rating for Employee ' || v_employee_id || ': ' || v_ratings(1));
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
```

D. Exception handling

Implicit Exception Handling:

1. Implicit Division by Zero Exception:

```
DECLARE
  v_dividend NUMBER := 10;
  v_divisor NUMBER := 0;
  v_result NUMBER;
BEGIN
  v_result := v_dividend / v_divisor;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: Division by zero');
END;
```

2. Implicit NO_DATA_FOUND Exception:

```
DECLARE
  v_employee_id employees.employee_id%TYPE := 9999;
  v_employee_name employees.first_name%TYPE;
BEGIN
  SELECT first_name INTO v_employee_name
  FROM employees
  WHERE employee_id = v_employee_id;

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error: Employee not found');
END;
```

3. Implicit Too_Many_Rows Exception:

```
DECLARE
  v_employee_name employees.first_name%TYPE;
BEGIN
  SELECT first_name INTO v_employee_name
  FROM employees
  WHERE department_id = 110;

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Error: Multiple employees found');
END;
```

Explicit Exception Handling:

1. Explicit Invalid_Number Exception:

```
DECLARE
  v_rating VARCHAR2(10) := 'ABC';
  v_skill_rating NUMBER;
BEGIN
  BEGIN
    v_skill_rating := TO_NUMBER(v_rating);
  EXCEPTION
    WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('Error: Invalid number format');
  END;
END;
```

2. Explicit Value_Error Exception:

```
DECLARE
  v_hours_worked NUMBER := -10;
BEGIN
  IF v_hours_worked < 0 THEN
    RAISE VALUE_ERROR;
  END IF;
EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Error: Invalid value');
END;
```

3. Explicit Too_Many_Days Exception:

```
DECLARE
  v_leave_days NUMBER := 40;
  v_max_leave_days NUMBER := 30;
BEGIN
  IF v_leave_days > v_max_leave_days THEN
    RAISE_APPLICATION_ERROR(-20001, 'Error: Number of leave days exceeds the maximum allowed');
  END IF;
EXCEPTION
```

```
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

E. Cursor management

Implicit Cursor without Parameters

```
DECLARE
  CURSOR c_employees IS
    SELECT first_name, last_name
    FROM employees;

  v_employee_first_name employees.first_name%TYPE;
  v_employee_last_name employees.last_name%TYPE;
BEGIN
  OPEN c_employees;

  LOOP
    FETCH c_employees INTO v_employee_first_name, v_employee_last_name;
    EXIT WHEN c_employees%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE
('Employee Name: ' || v_employee_first_name || ' ' || v_employee_last_name);
  END LOOP;

  CLOSE c_employees;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    CLOSE c_employees;
END;
```

Explicit Cursor with Parameters

```
DECLARE
  CURSOR c_employee_attendance (v_employee_id employees.employee_id%TYPE) IS
    SELECT attendance_date, hours_worked
    FROM attendance
    WHERE employee_id = v_employee_id;

  v_employee_id employees.employee_id%TYPE := 1001;
  v_attendance_date attendance.attendance_date%TYPE;
  v_hours_worked attendance.hours_worked%TYPE;
BEGIN
  OPEN c_employee_attendance(v_employee_id);
  LOOP
    FETCH c_employee_attendance INTO v_attendance_date, v_hours_worked;
    EXIT WHEN c_employee_attendance%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE
('Attendance for Employee ' || v_employee_id || ' on ' || v_attendance_date ||
 ': ' || v_hours_worked || ' hours');
  END LOOP;

  CLOSE c_employee_attendance;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    CLOSE c_employee_attendance;
END;
```

F. Functions, procedures, inclusion in packages

Functions:

1. Function to calculate the total hours worked by an employee on a given date:

```
CREATE OR REPLACE FUNCTION calculate_hours_worked(
  p_employee_id IN employees.employee_id%TYPE,
  p_attendance_date IN attendance.attendance_date%TYPE
)
RETURN NUMBER
IS
  v_total_hours NUMBER;
BEGIN
  SELECT SUM(hours_worked)
  INTO v_total_hours
  FROM attendance
  WHERE employee_id = p_employee_id
    AND attendance_date = p_attendance_date;

  RETURN v_total_hours;
END;
/

DECLARE
  v_employee_id employees.employee_id%TYPE := 123;
  v_attendance_date attendance.attendance_date%TYPE := DATE '2023-05-15';
  v_hours_worked NUMBER;
BEGIN
  v_hours_worked := calculate_hours_worked(v_employee_id, v_attendance_date);

  DBMS_OUTPUT.PUT_LINE('Total Hours Worked: ' || v_hours_worked);
END;
```

```
/
```

2. Function to calculate the average hours worked by an employee in a given month:

```
CREATE OR REPLACE FUNCTION calculate_average_hours_worked(
  p_employee_id IN employees.employee_id%TYPE,
  p_month IN NUMBER,
  p_year IN NUMBER
)
RETURN NUMBER
IS
  v_average_hours NUMBER;
BEGIN
  SELECT AVG(hours_worked)
  INTO v_average_hours
  FROM attendance
  WHERE employee_id = p_employee_id
    AND EXTRACT(MONTH FROM attendance_date) = p_month
    AND EXTRACT(YEAR FROM attendance_date) = p_year;

  RETURN v_average_hours;
END;
/

DECLARE
  v_employee_id employees.employee_id%TYPE := 123;
  v_month NUMBER := 5; -- Provide the desired month
  v_year NUMBER := 2023; -- Provide the desired year
  v_average_hours NUMBER;
BEGIN
  v_average_hours := calculate_average_hours_worked(v_employee_id, v_month, v_year);

  DBMS_OUTPUT.PUT_LINE('Average Hours Worked: ' || v_average_hours);
END;
/
```

3. Function to check if an employee has perfect attendance for a given year:

```
CREATE OR REPLACE FUNCTION check_perfect_attendance(
  p_employee_id IN employees.employee_id%TYPE,
  p_year IN NUMBER
)
RETURN VARCHAR2
IS
  v_attendance_count NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO v_attendance_count
  FROM attendance
  WHERE employee_id = p_employee_id
    AND EXTRACT(YEAR FROM attendance_date) = p_year;

  IF v_attendance_count = 365 OR v_attendance_count = 366 THEN
    RETURN 'Yes'; -- Perfect attendance
  ELSE
    RETURN 'No'; -- Not perfect attendance
  END IF;
END;
/

DECLARE
  v_employee_id employees.employee_id%TYPE := 123; -- Provide the desired employee ID
  v_year NUMBER := 2023; -- Provide the desired year
  v_attendance_status VARCHAR2(3);
BEGIN
  v_attendance_status := check_perfect_attendance(v_employee_id, v_year);

  DBMS_OUTPUT.PUT_LINE('Perfect Attendance: ' || v_attendance_status);
END;
/
```

Procedures:

1. Procedure to mark an employee as present for a given date:

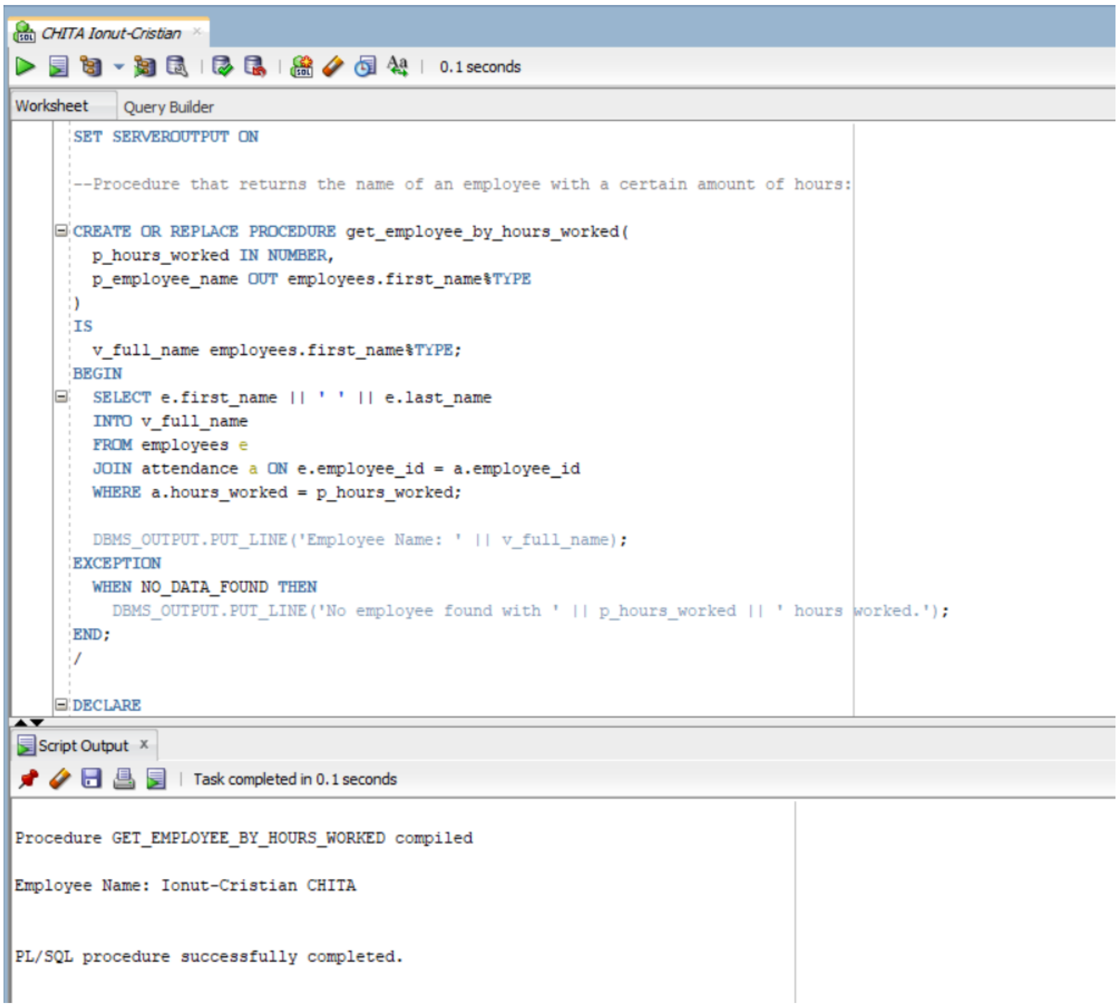
```
CREATE OR REPLACE PROCEDURE mark_employee_present(
  p_employee_id IN employees.employee_id%TYPE,
  p_attendance_date IN attendance.attendance_date%TYPE
)
IS
BEGIN
  INSERT INTO attendance (employee_id, attendance_date, hours_worked)
  VALUES (p_employee_id, p_attendance_date, 8); -- Assuming 8 hours for a full day

  COMMIT;
END;
/
```

2. Procedure to adjust hours worked for an employee on a specific date:

```
CREATE OR REPLACE PROCEDURE adjust_hours_worked(
  p_employee_id IN employees.employee_id%TYPE,
  p_attendance_date IN attendance.attendance_date%TYPE,
  p_adjustment_hours IN NUMBER
)
IS
BEGIN
  UPDATE attendance
  SET hours_worked = hours_worked + p_adjustment_hours
  WHERE employee_id = p_employee_id
    AND attendance_date = p_attendance_date;

  COMMIT;
END;
/
```

3. Procedure that returns the name of an employee with a certain amount of hours:

```
CREATE OR REPLACE PROCEDURE get_employee_by_hours_worked(
  p_hours_worked IN NUMBER,
  p_employee_name OUT employees.first_name%TYPE
)
IS
  v_full_name employees.first_name%TYPE;
BEGIN
  SELECT e.first_name || ' ' || e.last_name
  INTO v_full_name
  FROM employees e
  JOIN attendance a ON e.employee_id = a.employee_id
  WHERE a.hours_worked = p_hours_worked;

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_full_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No employee found with ' || p_hours_worked || ' hours worked.');
END;
/

DECLARE
  v_employee_name employees.first_name%TYPE;
BEGIN
  get_employee_by_hours_worked(30, v_employee_name);
END;
/
```


CHITA Ionut-Cristian

0.1 seconds

Worksheet Query Builder

```
SET SERVEROUTPUT ON

--Procedure that returns the name of an employee with a certain amount of hours:

CREATE OR REPLACE PROCEDURE get_employee_by_hours_worked(
  p_hours_worked IN NUMBER,
  p_employee_name OUT employees.first_name%TYPE
)
IS
  v_full_name employees.first_name%TYPE;
BEGIN
  SELECT e.first_name || ' ' || e.last_name
  INTO v_full_name
  FROM employees e
  JOIN attendance a ON e.employee_id = a.employee_id
  WHERE a.hours_worked = p_hours_worked;

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_full_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No employee found with ' || p_hours_worked || ' hours worked.');
END;
/

DECLARE
```

Script Output

Task completed in 0.1 seconds

```
Procedure GET_EMPLOYEE_BY_HOURS_WORKED compiled

Employee Name: Ionut-Cristian CHITA


PL/SQL procedure successfully completed.
```

Package:

```
CREATE OR REPLACE PACKAGE employee_management_pkg IS

  -- Function to calculate the total hours worked by an employee on a given date
  FUNCTION calculate_hours_worked(
    p_employee_id IN employees.employee_id%TYPE,
    p_attendance_date IN attendance.attendance_date%TYPE
  ) RETURN NUMBER;

  -- Procedure to mark an employee present for a given date
  PROCEDURE mark_employee_present(
    p_employee_id IN employees.employee_id%TYPE,
    p_attendance_date IN attendance.attendance_date%TYPE
  );
```

```
END employee_management_pkg;
/

CREATE OR REPLACE PACKAGE BODY employee_management_pkg IS

  -- Function to calculate the total hours worked by an employee on a given date
  FUNCTION calculate_hours_worked(
    p_employee_id IN employees.employee_id%TYPE,
    p_attendance_date IN attendance.attendance_date%TYPE
  ) RETURN NUMBER IS
    v_total_hours NUMBER;
  BEGIN
    SELECT SUM(hours_worked)
    INTO v_total_hours
    FROM attendance
    WHERE employee_id = p_employee_id
      AND attendance_date = p_attendance_date;

    RETURN v_total_hours;
  END calculate_hours_worked;

  -- Procedure to mark an employee present for a given date
  PROCEDURE mark_employee_present(
    p_employee_id IN employees.employee_id%TYPE,
    p_attendance_date IN attendance.attendance_date%TYPE
  ) IS
  BEGIN
    INSERT INTO attendance (employee_id, attendance_date, hours_worked)
    VALUES (p_employee_id, p_attendance_date, 8); -- Assuming 8 hours for a full day

    COMMIT;
  END mark_employee_present;

END employee_management_pkg;
/
```

G. Triggers at statement and row level

Statement-level Triggers

1. Before Insert on the Attendance Table:

```
CREATE OR REPLACE TRIGGER attendance_insert_trigger
BEFORE INSERT ON attendance
FOR EACH ROW
DECLARE
  total_hours NUMBER;
BEGIN
  SELECT SUM(hours_worked)
  INTO total_hours
  FROM attendance
  WHERE employee_id = :new.employee_id;

  IF total_hours + :new.hours_worked > 40 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Cannot insert attendance record. Maximum hours exceeded.');
  END IF;
END;
/
```

2. AFTER INSERT OR UPDATE OR DELETE:

```
CREATE OR REPLACE TRIGGER table_logging_trigger
AFTER INSERT OR UPDATE OR DELETE ON employees
DECLARE
  v_action VARCHAR2(10);
BEGIN
  IF INSERTING THEN
    v_action := 'INSERT';
  ELSIF UPDATING THEN
    v_action := 'UPDATE';
  ELSIF DELETING THEN
    v_action := 'DELETE';
  END IF;

  DBMS_OUTPUT.PUT_LINE('Table Employees was ' || v_action || 'd.');
END;
/
```

Row-level Trigger

1. Before Update on the Employees Table:

```
CREATE OR REPLACE TRIGGER employees_update_trigger
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
  IF :new.salary < :old.salary THEN
    RAISE_APPLICATION_ERROR(-20002, 'Cannot decrease the salary of an employee.');
  END IF;
END;
/
```

2. After Insert on the Employees Table:

```
CREATE OR REPLACE TRIGGER employees_insert_trigger
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO job_history(employee_id, start_date, end_date, job_id, department_id)
  VALUES (:new.employee_id, SYSDATE, NULL, :new.job_id, :new.department_id);
END;
/
```