Matei Ana Maria Valentina

Group 1067

DataBase Project

~ Theatre ~

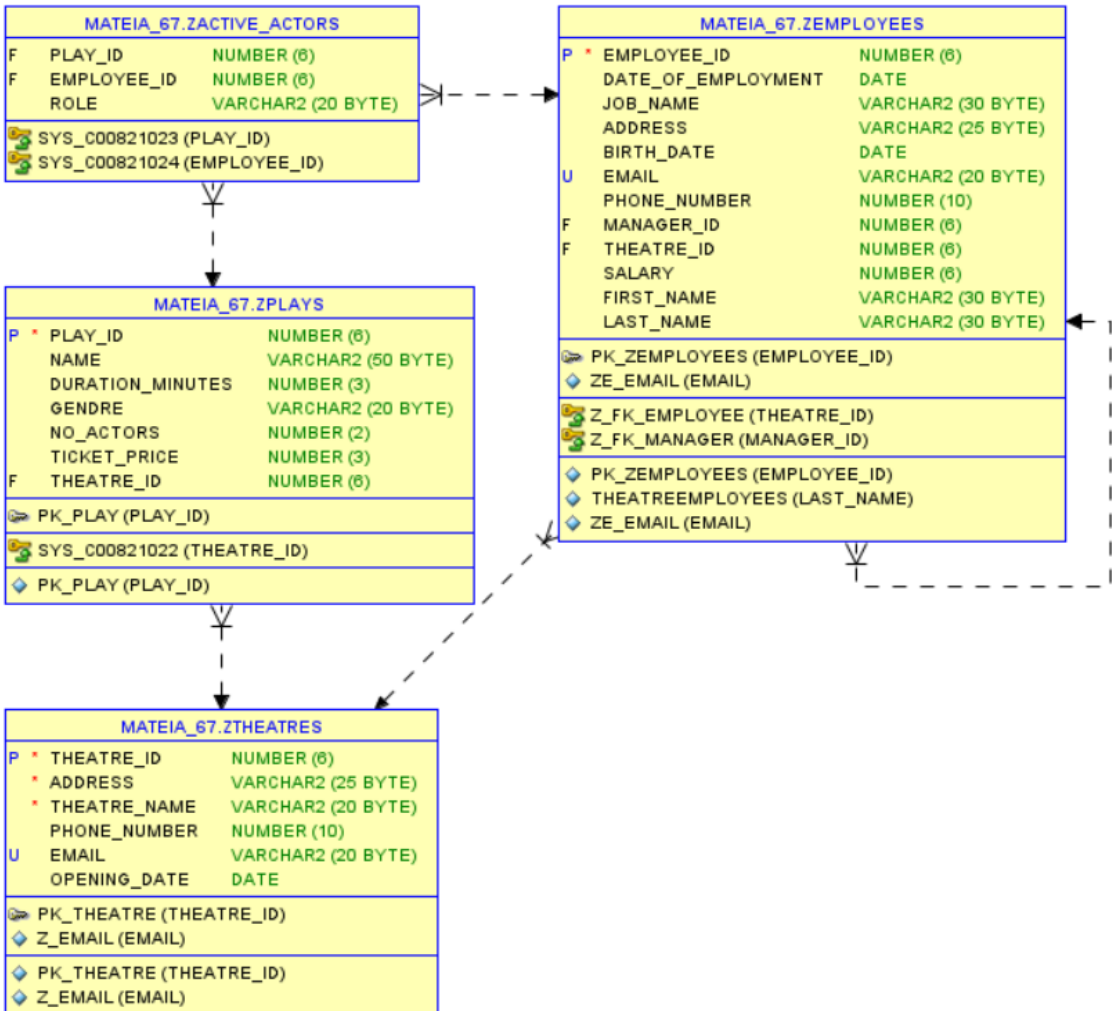
The database project that is created has as porpoise the storing of data for theatres, in such it explores the economic and managerial characteristics of a theatre, its employees, actors and plays.

I first started with the parent table “Theatres” which includes data such as the name, address and contact information for the Theatre in question. After the parent table was created, I could move on to the next few children tables.

My project also includes three additional tables that contain data for “Plays”, “Employees” and “Active actors” respectively. The “Plays” table has data about the gender, name, duration and other additional information relevant for both the theatre manager and for a potential costumer interested on a play offer.

The tables “Employees” and “Active actors” contain the personal information of the people employed in the theatre such as contact information, the name of the job, employee ID.

In the schema below it can be seen how the tables connect with each other and the overall columns and data types they contain:



I would want to mention that the database schema is taken form the project that I made in the first semester and it may have several columns changed due to the DDL statements.

1. Create a procedure which updates the salary of a given employee in which the salaries between 3500 and 4000 are updated with 20%, the salaries between 4000 and 4500 are updated with 15% and the salaries between 4500 and 5000 are updated with 5%, treat the exceptions if any.

```
CREATE OR REPLACE PROCEDURE update_salaries (v_id IN NUMBER)
IS
v_salary NUMBER;
BEGIN
  SELECT salary INTO v_salary
  FROM ZEmployees
  WHERE employee_id = v_id;
  DBMS_OUTPUT.PUT_LINE('The salary before the update is: ' || v_salary);
  UPDATE ZEmployees
  SET salary = CASE
    WHEN salary BETWEEN 3500 AND 4000 THEN salary * 1.2
    WHEN salary BETWEEN 4000 AND 4500 THEN salary * 1.15
    WHEN salary BETWEEN 4500 AND 5000 THEN salary * 1.05
    ELSE salary
  END
  WHERE employee_id = v_id;
  SELECT salary INTO v_salary
  FROM ZEmployees
  WHERE employee_id = v_id;
  DBMS_OUTPUT.PUT_LINE('Salaries updated successfully to: ' || v_salary );
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error updating salaries, no data was found: ' || SQLERRM);
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error updating salaries: ' || SQLERRM);
END;
/
```

2. Add the column “gender” to the ZActive_Actors table.

```
BEGIN
  EXECUTE IMMEDIATE 'ALTER TABLE ZActive_actors ADD (gender VARCHAR2(20))';
  DBMS_OUTPUT.PUT_LINE('Column "gendre" added successfully.');
END;
/
```

3. Display the highest salary of the employees and the job name in upper cases, treat any exception that might occur.

```
SET SERVEROUTPUT ON
DECLARE
  v_max_salary NUMBER;
  v_job VARCHAR2(30);
BEGIN
  SELECT MAX(salary) INTO v_max_salary FROM ZEmployees;

  SELECT UPPER(job_name) INTO v_job
  FROM ZEmployees
  WHERE salary = v_max_salary;

  DBMS_OUTPUT.PUT_LINE('Maximum Salary: ' || v_max_salary);
  DBMS_OUTPUT.PUT_LINE('Job (Uppercase): ' || v_job);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Error: Too many rows returned.');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error: No data found.');
  WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
END;
/
```

```
SET SERVEROUTPUT ON
DECLARE
  v_max_salary NUMBER;
  v_job VARCHAR2(30);
BEGIN
  SELECT MAX(salary) INTO v_max_salary FROM ZEmployees;

  SELECT UPPER(job_name) INTO v_job
  FROM ZEmployees
  WHERE salary = v_max_salary;

  DBMS_OUTPUT.PUT_LINE('Maximum Salary: ' || v_max_salary);
  DBMS_OUTPUT.PUT_LINE('Job (Uppercase): ' || v_job);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Error: Too many rows returned.');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Error: No data found.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
END;
/
```

Script Output

Task completed in 0.04 seconds

```
Error: Too many rows returned.


PL/SQL procedure successfully completed.
```

4. Display the theatres id and names and all of their employees. (Here the output displays my name)

```
SET SERVEROUTPUT ON
DECLARE
  j NUMBER := 0;
  CURSOR c_theatres IS
    SELECT theatre_id, theatre_name
    FROM ZTheatres;

  CURSOR c_employees (p_theatre_id NUMBER) IS
    SELECT *
    FROM ZEmployees
    WHERE theatre_id = p_theatre_id;

  v_theatre_id ZTheatres.theatre_id%TYPE;
  v_theatre_name ZTheatres.theatre_name%TYPE;
  v_employee ZEmployees%ROWTYPE;
BEGIN
  OPEN c_theatres;
  LOOP
    FETCH c_theatres INTO v_theatre_id, v_theatre_name;
    EXIT WHEN c_theatres%NOTFOUND;
    j:=0;
    DBMS_OUTPUT.PUT_LINE( 'Theatre ID: ' || v_theatre_id || ' Theatre name: ' ||
v_theatre_name);
    DBMS_OUTPUT.PUT_LINE(' ');

    OPEN c_employees(v_theatre_id);
```

```
    LOOP
    j:=j+1;
      FETCH c_employees INTO v_employee;
      IF c_employees%NOTFOUND AND j=1 THEN
      DBMS_OUTPUT.PUT_LINE('There are no employees');
       DBMS_OUTPUT.PUT_LINE(' ');
      END IF;
      EXIT WHEN c_employees%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE( j || ' ' || 'Employee ID: ' || v_employee.employee_ID || ' Name:
' || v_employee.first_name || ' ' || v_employee.last_name ||
      ' Job Name: ' || v_employee.job_name);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(' ');
    CLOSE c_employees;
  END LOOP;

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

```
Theatre ID: 2 Theatre name: Teatrul National

1 Employee ID: 23 Name: Ana Maria Matei Job Name: actor
2 Employee ID: 12 Name: Maricica Iliescu Job Name: manager
3 Employee ID: 7 Name: Daria Marin Job Name: actor
4 Employee ID: 6 Name: Mirela Dragomir Job Name: actor
5 Employee ID: 18 Name: Radu Sentil Job Name: staff
```

5. Display the name of the plays and their duration using a table by index

```
SET SERVEROUTPUT ON
DECLARE
  TYPE play_info IS TABLE OF VARCHAR(500) INDEX BY PLS_INTEGER;
  v_play_names play_info;
  v_play_durations play_info;
BEGIN
  FOR rec IN (SELECT play_id, name, duration_minutes FROM ZPlays)
  LOOP
    v_play_names(rec.play_id) := rec.name;
    v_play_durations(rec.play_id) := rec.duration_minutes;
  END LOOP;
  FOR i IN v_play_names.FIRST..v_play_names.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE('Play ID: ' || i);
    DBMS_OUTPUT.PUT_LINE('Play Name: ' || v_play_names(i));
    DBMS_OUTPUT.PUT_LINE('Duration (minutes): ' || v_play_durations(i));
    DBMS_OUTPUT.PUT_LINE('----------------------------');
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
END;
/
```

```
Play ID: 31
Play Name: Moris
Duration (minutes): 70
---------------------------
Play ID: 32
Error occurred: ORA-01403: no data found


PL/SQL procedure successfully completed.
```

6. Create a function that verifies is the actor is a lead actor or not based on their Id.

```
CREATE OR REPLACE FUNCTION IsLeadActor(p_actor_id NUMBER)
RETURN BOOLEAN
IS
  v_lead_actor ZActive_actors.role%TYPE;
BEGIN
  SELECT role INTO v_lead_actor
  FROM ZActive_Actors
  WHERE employee_id = p_actor_id AND role = 'lead actor';
  IF v_lead_actor IS NOT NULL THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END; /
```

7. Create a function that gives a 15% discount for children to any play.

```
CREATE OR REPLACE FUNCTION DiscountForChildren (p_play_name IN VARCHAR2)
RETURN NUMBER
IS
  v_ticket_price ZPlays.ticket_price%TYPE;
  v_discounted_price NUMBER;
BEGIN
  SELECT ticket_price INTO v_ticket_price
  FROM ZPlays
  WHERE name = p_play_name;

  v_discounted_price := v_ticket_price * 0.85;

  RETURN v_discounted_price;
END;
/
```

8. Create a procedure that displays the average number of minutes for a play and raises a user defined exception so that we would know if it was too long for a child to sit through.

```
CREATE OR REPLACE PROCEDURE DisplayAverageMinutes(p_play_name IN
VARCHAR2)
IS
  v_average_minutes NUMBER;
  v_play_minutes NUMBER;
  ex_play_too_long EXCEPTION;
  PRAGMA EXCEPTION_INIT(ex_play_too_long, -20001);
BEGIN
```

```
  SELECT AVG(duration_minutes) INTO v_average_minutes
  FROM ZPlays;

  SELECT duration_minutes INTO v_play_minutes
  FROM ZPlays
  WHERE name = p_play_name;

  DBMS_OUTPUT.PUT_LINE('The average number of minutes for plays is: ' ||
v_average_minutes);
  DBMS_OUTPUT.PUT_LINE('The number of minutes for play ' || p_play_name || ' is: ' ||
v_play_minutes);

  IF v_play_minutes > v_average_minutes THEN
    RAISE ex_play_too_long;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Play not found');
  WHEN ex_play_too_long THEN
    DBMS_OUTPUT.PUT_LINE('The play is too long for children to watch');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred');
END;
/
```

9. Create a trigger that displays what has been deleted from the table ZPlays

```
CREATE OR REPLACE TRIGGER ZPlays_Delete_Trigger
AFTER DELETE ON ZPlays
FOR EACH ROW
DECLARE
  v_play_name VARCHAR2(50);
BEGIN
  v_play_name := :OLD.name;
  DBMS_OUTPUT.PUT_LINE('A record with the play name "' || v_play_name || '" was deleted from ZPlays table.');
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/
```

10. Create a trigger that informs the user if the salary has been updated before the update

```
CREATE OR REPLACE TRIGGER Salary_Update_Trigger
BEFORE UPDATE ON ZEmployees
FOR EACH ROW
DECLARE
  v_old_salary NUMBER;
BEGIN
  v_old_salary := :OLD.salary;
  IF :NEW.salary <> v_old_salary THEN
    DBMS_OUTPUT.PUT_LINE('The salary for employee with ID ' || :NEW.employee_ID || ' is being updated.');
```

```
  END IF;
END;
/
```

11. Create a trigger that informs if an update or insert was made in ZEmployees

```
CREATE OR REPLACE TRIGGER Insert_Update_Trigger
AFTER INSERT OR UPDATE ON ZEmployees
BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('An INSERT operation was performed on ZEmployees table.');
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('An UPDATE operation was performed on ZEmployees
table.');
  END IF;
END;
/
```

12. Create a trigger that informs the user if a delete statement was made in the ZActive_Actors table.

```
CREATE OR REPLACE TRIGGER Delete_from_ZTable
AFTER DELETE ON ZActive_Actors
BEGIN
  DBMS_OUTPUT.PUT_LINE('A DELETE operation was performed on ZActive_Actors
table.');
END;
/
```

13. Create a package that has a function that returns the phone number of a theatre and a procedure that tells you how long it has been open, with an exception that says "Is a historic monument" if it has over 30 years old.

```
CREATE OR REPLACE PACKAGE TheatreInfo
AS
  FUNCTION GetTheatrePhoneNumber(p_theatre_name IN VARCHAR2)
  RETURN NUMBER;
  PROCEDURE CalculateTheatreAge(p_theatre_id IN NUMBER);
END;
/

CREATE OR REPLACE PACKAGE BODY TheatreInfo
AS
  FUNCTION GetTheatrePhoneNumber( p_theatre_name IN VARCHAR2)
  RETURN NUMBER
  IS
    v_phone_number NUMBER;
  BEGIN
    SELECT phone_number INTO v_phone_number
    FROM ZTheatres
    WHERE theatre_name = p_theatre_name;
    RETURN v_phone_number;
  END;

  PROCEDURE CalculateTheatreAge(p_theatre_id IN NUMBER)
  IS
    v_opening_date DATE;
    v_years_open NUMBER;
```

```
    v_exception_msg VARCHAR2(100) := 'The building is a historic monument.';
  BEGIN
    SELECT opening_date INTO v_opening_date
    FROM ZTheatres
    WHERE theatre_id = p_theatre_id;

    v_years_open := TRUNC(MONTHS_BETWEEN(SYSDATE, v_opening_date) / 12);

    IF v_years_open > 30 THEN
      RAISE_APPLICATION_ERROR(-20001, v_exception_msg);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Theatre has been open for ' || v_years_open || ' years.');
    END IF;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Theatre with ID ' || p_theatre_id || ' does not exist.');
  END;
END;
/
```