| **Profesor:** | **Antonio Luis Pantoja Valero** | **Grupo:** | **5** |
| --- | --- | --- | --- |
| **Alumno/a:** | **Ángela Morales Valverde** | **NIA:** | **100432200** |
| **Alumno/a:** | **Samuel Fernández Fernández** | **NIA:** | **100432070** |
| **Alumno/a:** | **David Merino Hernández** | **NIA:** | **100432028** |

# ÍNDICE

[**ÍNDICE 1**](#_uwvbgb9jgqjg)

[**1. Introducción 3**](#_t6c6reggffb0)

[**2. Consultas 3**](#)

[CONSULTA 1 3](#_u8p57bfjsde8)

[a) su diseño en álgebra relacional 3](#_w2bcyupaxebp)

[b) su implementación en SQL 4](#_gv9skvdav5br)

[c) las pruebas realizadas para demostrar que funciona correctamente 5](#_8t325uo1yi12)

[CONSULTA 2 6](#_h1zj048tpseq)

[d) su diseño en álgebra relacional 7](#_vrwykt5vtn58)

[e) su implementación en SQL 8](#_yianybvynb4z)

[f) las pruebas realizadas para demostrar que funciona correctamente 9](#_x24huiia6jyc)

[**3. Paquete 10**](#)

[OPERATIVIDAD 1.2.1 10](#_8m3yisuqvk78)

[a) su diseño 10](#_oqh1rm8p5ts1)

[b) su implementación en SQL 11](#_vdh5okjuzsc7)

[c) pruebas 12](#_dx89v3h6um5m)

[OPERATIVIDAD 1.2.2 13](#_8n4qamg25e2r)

[d) su diseño 13](#_m3r8zsgfs6uu)

[e) su implementación en SQL 15](#_g1xxq06lnbqe)

[f) pruebas 16](#_jbgin3yxr8d0)

[OPERATIVIDAD 1.2.3 17](#_6hj3utzbopmk)

[g) su diseño 17](#_335g7xidsmsl)

[h) su implementación en SQL 18](#_syj3z6rxhgjb)

[i) pruebas 19](#_lxu64bklwpor)

[OPERATIVIDAD 1.2.4 19](#_g31l09a8m71j)

[j) su diseño 20](#_3g86mscrgur8)

[k) su implementación en SQL 20](#_r4n2j0gn9obv)

[l) pruebas 24](#_qw7n9vf0qhn7)

[PAQUETE MELOPACK 26](#_8jlk995hhg6k)

[**4. Diseño externo 32**](#)

[VISTA 1 32](#_ghl2egsmfj1h)

[a. su diseño en álgebra relacional 32](#_j781euj34qc8)

[b. su implementación en SQL 32](#_xpzq9o4sxp4)

[c. Pruebas 33](#_do2r8uobhyfp)

[VISTA 2 33](#_17uemoqxb7k5)

[d. su diseño en álgebra relacional 34](#_urkktl433hyc)

[e. su implementación en SQL 34](#_mhrs5iagutjb)

[f. Pruebas 35](#_u44kqxnoyyzk)

[VISTA 3 35](#_9frv9atcs7a1)

[g. su diseño en álgebra relacional 35](#_nd7lotasksnq)

[h. su implementación en SQL 36](#_xxbixerdxgk1)

[i. Pruebas 38](#_lzbsggcm7o8v)

[**5. Disparadores 40**](#)

[DISPARADOR A 40](#_g9bi2z4m66xq)

[a) Descripción del diseño 40](#_xo7ygitrxcpt)

[b) Código (en SQL) 41](#_1zukyj1f973e)

[c) Pruebas 42](#_qipkcjm0nayf)

[DISPARADOR B 42](#_lqqdw4j83j3n)

[d) Descripción del diseño 42](#_qic4clmo5cv5)

[e) Código (en SQL) 43](#_e0g716u8kcyf)

[f) Pruebas 43](#_3d1c7fpcqm28)

[DISPARADOR C 44](#_7norr2i9mt33)

[g) Descripción del diseño 44](#_svx5gk649zhp)

[h) Código (en SQL) 45](#_2rao28y0exg3)

[i) Pruebas 45](#_ef7wu4tnkez4)

[**6. Conclusiones 46**](#)

# 

# Introducción

Nuestro punto de partida se basa en las tablas e inserciones de datos que se nos han proporcionado como solución de la primera práctica, teniéndolos que tratar de distintas formas dichos datos en esta práctica actual.

Este trabajo se compone de cuatro partes principalmente siendo estas las consultas, la creación de un paquete de funciones y procedimientos, las vistas sobre los datos que hay almacenados en las tablas y la elaboración de disparadores que controlen las acciones de inserción y borrado dentro de las tablas que así se nos indica.

Hemos organizado esta memoria en el mismo orden que van esos apartados y teniendo el objetivo de poder abarcar la mayor parte de la operatividad de la práctica, tratando cada apartado por separado y apoyándonos en secciones que ya hayamos hecho para la resolución de las siguientes.

# Consultas

**Para cada consulta se debe incluir una subsección, en la que se describirán:**

## CONSULTA 1

**Los intérpretes que cantan canciones escritas por ellos mismos (cuya autoría corresponde a cualquiera de los miembros de la banda o al solista, según sea el caso). La consulta proporcionará el nombre del intérprete, el porcentaje de pistas (tracks) grabadas que son canciones propias, y el porcentaje de interpretaciones en conciertos que son canciones propias.**

### su diseño en álgebra relacional

En la siguiente tabla detallamos los pasos que hemos seguido para desarrollar el álgebra relacional de estas consultas.

| I ≡ PERFORMERS (intérpretes) |
| --- |
| C ≡ CONCERTS (conciertos) |
| P ≡ PERFORMANCES (interpretaciones) |
| S ≡ SONGS (canciones) |
| T ≡ TRACKS (pistas) |
| P ≡ PERFORMANCES (interpretaciones) |
| **SQ1** ≡ **ㅠname** ( I ) |
| **SQ2** ≡ **ㅠ C.performer, SQ1.name ( SQ1 \* C.performer C )** |
| **SQ3** ≡ **ㅠ P.songwriter, P.songtitle, SQ2.name ( SQ2 Ө C.Performer =P.performer P )** |
| **SQ4** ≡ **ㅠ SQ3.name, S.writer, S.title S.cowriter ( SQ3 Ө SQ3.songwriter = S.writer v SQ3.songwriter = S.cowriter ^ SQ3.songtitle = S.title S)** |
| **SQ5** ≡ **ㅠ SQ4.name( SQ4 Ө SQ4.title = T.title ^ SQ4.writer = T.writer T)** |
| **SQ6** ≡ **ㅠ SQ5.name, porcentaje\_pp, porcentaje\_icp  ( σ porcentaje\_pp = count(sq5) / count(\*)from T \* 100 SQ5, σ porcentaje\_icp = count(sq5) / count(\*)from P \* 100 SQ5)** |
| **RESULTADO** ≡ **ㅠ SQ6.name, SQ6.porcentaje\_pp, SQ6.porcentaje\_icp SQ6** |

### su implementación en SQL

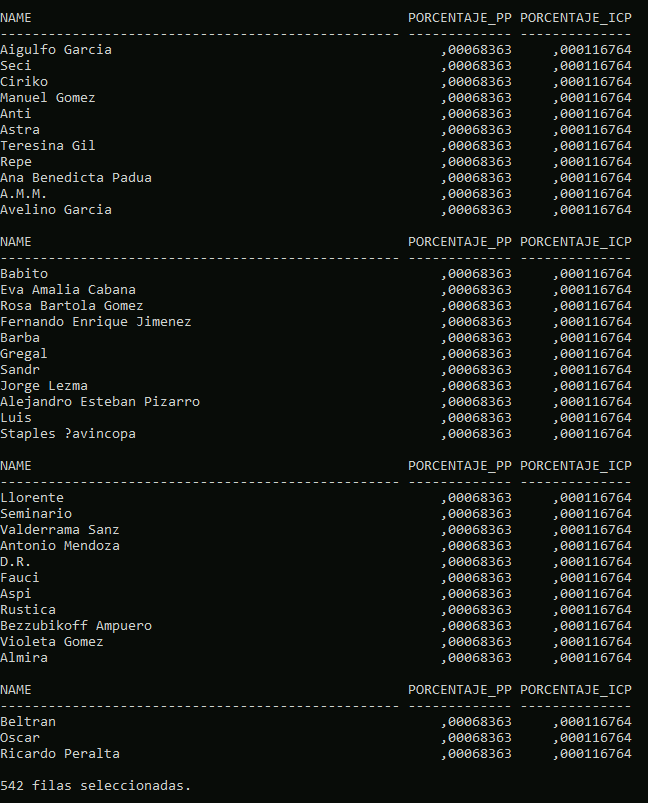
| **WITH interpretes AS (SELECT DISTINCT name FROM PERFORMERS), conciertos\_interpretes AS (SELECT DISTINCT C.performer, I.name FROM CONCERTS C INNER JOIN interpretes I ON (C.performer = I.name)), performer\_performances AS (SELECT DISTINCT CI.name, P.songwriter FROM PERFORMANCES P INNER JOIN conciertos\_interpretes CI ON (P.performer=CI.performer)), interpreted\_songs AS (SELECT DISTINCT PP.name, S.writer, S.cowriter FROM SONGS S INNER JOIN performer\_performances PP ON (PP.songwriter=S.writer)), tracked\_songs AS (SELECT DISTINCT INS.name FROM TRACKS T INNER JOIN interpreted\_songs INS ON(INS.writer = T.writer OR INS.cowriter = T.writer)), porcentajes\_artista AS (SELECT DISTINCT TS.name, (COUNT(\*) / (SELECT COUNT(\*) FROM TRACKS)\* 100) AS porcentaje\_pp, (COUNT(\*)/ (SELECT COUNT(\*) FROM PERFORMANCES)\*100) AS porcentaje\_icp FROM tracked\_songs TS GROUP BY TS.name)** SELECT DISTINCT name, porcentaje\_pp, porcentaje\_icp FROM **porcentajes\_artista** ; |
| --- |

### las pruebas realizadas para demostrar que funciona correctamente

Un ejemplo de la salida que tiene que salir al introducir el fragmento de código que se ha especificado en el apartado anterior sería el que se muestra en la siguiente imagen, donde se ven los tres campos que se nos ha pedido que devolvamos en la consulta.

Como se puede apreciar en la imagen que se encuentra a continuación, los porcentajes que esta consulta muestra para todos los intérpretes tienen el mismo valor. Sabemos que no todos los intérpretes tienen el mismo número de tracks, por ejemplo, por lo que este porcentaje debería ser distinto para cada intérprete. Pese a saberlo, nos ha sido imposible solucionarlo dado al poco tiempo del que hemos podido disponer para la realización de la práctica.

Un ejemplo de la tabla que sacaría esta consulta es la que se muestra a continuación en la imagen.



## CONSULTA 2

**Para los diez intérpretes con mayor porcentaje de interpretaciones grabadas (canciones interpretadas en conciertos que ese mismo grupo ha grabado anteriormente), proporcionar la edad media de las canciones interpretadas (siendo la edad el tiempo transcurrido entre la grabación y la interpretación). La salida debe incluir el nombre del intérprete, el porcentaje de interpretaciones grabadas, y la edad media de las canciones (en años, meses, y días).**

### su diseño en álgebra relacional

Ya que parte del diseño de esta consulta viene de la anterior, hemos utilizado algunas subquerys adaptándolas a los datos que buscamos en esta segunda consulta.

| I ≡ PERFORMERS (intérpretes) |
| --- |
| C ≡ CONCERTS (conciertos) |
| P ≡ PERFORMANCES (interpretaciones) |
| S ≡ SONGS (canciones) |
| T ≡ TRACKS (pistas) |
| P ≡ PERFORMANCES (interpretaciones) |
| **SQ1** ≡ **ㅠname** ( I ) |
| **SQ2** ≡ **ㅠ C.performer, SQ1.name ( SQ1 \* C.performer C )** |
| **SQ3** ≡ **ㅠ P.songwriter, P.songtitle, P.when, SQ2.name ( SQ2 Ө C.Performer =P.performer P )** |
| **SQ4** ≡ **ㅠ SQ3.name, SQ3.when, S.writer, S.title S.cowriter ( SQ3 Ө SQ3.songwriter = S.writer v SQ3.songwriter = S.cowriter ^ SQ3.songtitle = S.title S)** |
| **SQ5** ≡ **ㅠ SQ4.name, SQ4.when, T.recording\_date ( SQ4 Ө SQ4.title = T.title ^ SQ4.writer = T.writer T)** |
| **SQ6** ≡ **ㅠ SQ5.name, porcentaje\_pp, porcentaje\_icp, SQ5.when, SQ5.recording\_date (σ porcentaje\_pp = count(sq5) / count(\*)from T \* 100 SQ5, σ porcentaje\_icp = count(sq5) / count(\*)from P \* 100 SQ5)** |
| **SQ7** ≡ **σrownum ≤ 10 〒porcentajes\_artista (SQ6)** |
| **SQ8** ≡ **ㅠ** AVG(**SQ6.when -****SQ6.recording\_date) (SQ7)** |
| **SQ9** ≡ SQ8 / 365 |
| **SQ10** ≡ SQ8 \* 12 |
| **SQ11** ≡ **ㅠ SQ6.porcentaje\_icp SQ6** |
| **RESULTADO** ≡ **ㅠ SQ6.name as ‘Nombre del interprete’, SQ11, SQ9, SQ10, SQ8** |

### su implementación en SQL

| **WITH interpretes AS (SELECT DISTINCT name FROM PERFORMERS),**  **conciertos\_interpretes AS (SELECT DISTINCT C.performer, I.name FROM CONCERTS C INNER JOIN interpretes I ON (C.performer = I.name)),**  **performer\_performances AS (SELECT DISTINCT CI.name, P.songwriter, P.when FROM PERFORMANCES P INNER JOIN conciertos\_interpretes CI ON (P.performer=CI.performer)),**  **interpreted\_songs AS (SELECT DISTINCT PP.name, PP.when, S.writer, S.cowriter FROM SONGS S INNER JOIN performer\_performances PP ON (PP.songwriter=S.writer)), tracked\_songs AS (SELECT DISTINCT INS.name, INS.when, T.rec\_date FROM TRACKS T INNER JOIN interpreted\_songs INS ON(INS.writer = T.writer OR INS.cowriter = T.writer)), porcentajes\_artista AS (SELECT DISTINCT TS.name, (COUNT(\*) / (SELECT COUNT(\*) FROM TRACKS)\* 100) AS porcentaje\_pp, (COUNT(\*)/ (SELECT COUNT(\*) FROM PERFORMANCES)\*100) AS porcentaje\_icp FROM tracked\_songs TS GROUP BY TS.name), fechas AS (SELECT DISTINCT TS.name, TS.when, TS.rec\_date FROM tracked\_songs TS), total AS (SELECT PA.name, PA.porcentaje\_pp, PA.porcentaje\_icp, F.when, F.rec\_date FROM porcentajes\_artista PA LEFT JOIN fechas F ON (PA.name = F.name)) SELECT DISTINCT name, porcentaje\_pp, porcentaje\_icp, when, rec\_date FROM total;** |
| --- |

### las pruebas realizadas para demostrar que funciona correctamente

Esta consulta muestra las querys de la consulta primera adaptadas a los datos que necesitamos en esta consulta.

Hemos tenido problemas a la hora de ejecución de esta segunda consulta y no hemos podido terminar su implementación por lo que no podemos mostrar pruebas sobre ella.

# Paquete

**Incluye una introducción con la estructura del paquete, y una subsección por cada procedimiento o función que incluya. Para cada procedimiento, debes exponer:**

* **su diseño (entradas, salidas, lógica del bloque principal), y en caso de haber necesitado hacer uso de elementos auxiliares (consultas, vistas, otros procedimientos/funciones…) también se debe incluir el diseño de estos (a menos que sean consultas triviales)**
* **su implementación en SQL**
* **pruebas**

## OPERATIVIDAD 1.2.1

**Variable “intérprete actual” y procedimiento que permita asignarle un valor dado (parám.).**

### su diseño

Declaramos una variable curr\_user para sacar el intérprete actual y una función getter siendo un VARCHAR2(50), como tiene el nombre de los intérpretes.

Para poder asignar un valor dado, en este caso hemos decidido que sea el nombre, necesitamos seguir el siguiente diseño:

| found = 0  for fila in CONCERTS:  if (CONCERTS.performer==performer\_name):  found++  break;  if found>= 1:  curr\_user = performer\_name  print("ASIGNACIÓN HECHA DE FORMA CORRECTA")  else:  print("SE HA PRODUCIDO UN ERROR") |
| --- |

De esta forma; si encontramos el usuario que nos piden, podemos darle un valor por defecto que se nos indique.

### su implementación en SQL

Para implementar este diseño hemos creado dos funciones. La función get\_curr\_user no recibe nada y lo que hace es inicializar/crear la variable del intérprete actual. Por otra parte, la función set\_curr\_user recibe el nombre del intérprete que quieres fijar como el actual y te lo devuelve dentro de la variable curr\_user para poderlo usar posteriormente en otras implementaciones en las que sea necesario. Esto lo hace si la función detecta que el intérprete que le estas dando como argumento existe dentro de la tabla de PERFORMERS ya que en caso de que no existiese, devolvería un mensaje mostrando que no se ha podido establecer y acabando así esta función, sin devolver ningún intérprete actual.

| create or replace FUNCTION get\_curr\_user RETURN VARCHAR2 IS  curr\_user VARCHAR2(100);  BEGIN  RETURN curr\_user;  END;  create or replace FUNCTION set\_curr\_user(  performer\_name IN PERFORMERS.name%TYPE  ) RETURN VARCHAR2 IS  curr\_user VARCHAR2(100);  found NUMBER(4) DEFAULT 0;  result VARCHAR2(100);  BEGIN  curr\_user := get\_curr\_user();  DBMS\_OUTPUT.PUT\_LINE('El usuario actual es ' || curr\_user);  SELECT count('x')  INTO found  FROM PERFORMERS  WHERE PERFORMERS.name = performer\_name;  IF found >= 1 THEN  curr\_user := performer\_name;  DBMS\_OUTPUT.PUT\_LINE('El usuario actual es ' || curr\_user);  DBMS\_OUTPUT.PUT\_LINE('OKAY CURR\_USER ESTABLECIDO');  ELSE  DBMS\_OUTPUT.PUT\_LINE('ERROR CURR\_USER NO SE HA PODIDO ESTABLECER');  END IF;  RETURN curr\_user;  END; |
| --- |

### pruebas

Probamos este procedimiento con un nombre conocido y otro desconocido y vemos que nos sale.

| Poniendo este código en el que seleccionamos un performer dentro de PERFORMERS que sí que existe obtenemos el resultado que se muestra:    RESULTADO    En cambio si introducimos un performer que no esté dentro de los que están dentro de la tabla PERFORMER (que no exista en la base de datos), obtenemos el siguiente resultado.    RESULTADO |
| --- |

## OPERATIVIDAD 1.2.2

**Insertar nuevo álbum del intérprete actual, con una canción (track); todas las filas referenciadas deberán existir previamente, y sólo se insertan el álbum y la pista (track); los datos necesarios (menos el intérprete, que es el actual) se proporcionan por parámetros; si el álbum (pair) ya existiera, se inserta sólo el track dado.**

### su diseño

Una posible implementación del diseño de este procedimiento en python sería el que se muestra a continuación aunque se ve representado con variaciones y ejemplos para mostrar la estructura del procedimiento que desarrollaremos posteriormente en sql.

| import datetime  def insertar\_album\_cancion(p\_album\_nombre, p\_cancion\_nombre, p\_cancion\_writer, p\_cancion\_recdate,p\_cancion\_studio, p\_cancion\_engineer, p\_cancion\_duration, p\_cancion\_sequ, p\_peformer, p\_pair, p\_format, p\_publisher, p\_manager):    v\_interprete\_actual = 'user' # Asignar valor a v\_interprete\_actual    # Consulta para obtener v\_album\_id  v\_album\_id = None  albums = [{'pair': '001', 'title': 'Album1'}, {'pair': '002', 'title': 'Album2'}, {'pair': '003', 'title': 'Album3'}]  for album in albums:  if album['pair'] == p\_pair:  v\_album\_id = album['pair']  break    # Si v\_album\_id es None, insertar un nuevo álbum  if v\_album\_id is None:  v\_album\_id = '004' # Asignar nuevo valor a v\_album\_id  album = {'pair': v\_album\_id, 'title': p\_album\_nombre, 'performer': v\_interprete\_actual, 'rel\_date': datetime.datetime.now().strftime('%Y'), 'format': p\_format, 'publisher': p\_publisher, 'manager': p\_manager} # Crear nuevo álbum  albums.append(album) # Agregar nuevo álbum a la lista    # Insertar nueva canción en TRACKS  track = {'pair': v\_album\_id, 'title': p\_cancion\_nombre, 'duration': p\_cancion\_duration, 'rec\_date': p\_cancion\_recdate, 'studio': p\_cancion\_studio, 'engineer': p\_cancion\_engineer, 'sequ': p\_cancion\_sequ, 'writer': p\_cancion\_writer} # Crear nueva canción  tracks = [{'pair': '001', 'title': 'Cancion1', 'duration': 3.5, 'rec\_date': datetime.datetime(2022, 3, 27), 'studio': 'Estudio1', 'engineer': 'Engineer1', 'sequ': 1, 'writer': 'Writer1'},  {'pair': '001', 'title': 'Cancion2', 'duration': 4.2, 'rec\_date': datetime.datetime(2022, 3, 28), 'studio': 'Estudio2', 'engineer': 'Engineer2', 'sequ': 2, 'writer': 'Writer2'},  {'pair': '002', 'title': 'Cancion3', 'duration': 3.7, 'rec\_date': datetime.datetime(2022, 3, 29), 'studio': 'Estudio3', 'engineer': 'Engineer3', 'sequ': 1, 'writer': 'Writer3'}] # Ejemplo de datos  tracks.append(track) # Agregar nueva canción a la lista    print('Nuevo álbum y canción agregados exitosamente') |
| --- |

### su implementación en SQL

| CREATE OR REPLACE PROCEDURE insertar\_album\_cancion(  p\_album\_nombre IN VARCHAR2,  p\_cancion\_nombre IN VARCHAR2,  p\_cancion\_writer IN VARCHAR2,  p\_cancion\_recdate IN DATE,  p\_cancion\_studio IN VARCHAR2,  p\_cancion\_engineer IN VARCHAR2,  p\_cancion\_duration IN NUMBER,  p\_cancion\_sequ IN NUMBER,  p\_pair IN VARCHAR2,  p\_format IN VARCHAR2,  p\_publisher IN VARCHAR2,  p\_manager IN VARCHAR2)  IS  v\_interprete\_actual VARCHAR2(50);  v\_album\_id VARCHAR2(50);  curr\_user VARCHAR2(50),  BEGIN  curr\_user := get\_curr\_user\_name();  SELECT user INTO v\_interprete\_actual FROM DUAL;    SELECT pair INTO v\_album\_id FROM ALBUMS WHERE pair = p\_pair;    IF v\_album\_id IS NULL THEN  INSERT INTO ALBUMS(title, performer, rel\_date, pair, format, publisher, manager)  VALUES(p\_album\_nombre, v\_interprete\_actual, TO\_CHAR(SYSDATE, 'YYYY'), p\_pair, p\_format, p\_publisher, p\_manager);    SELECT MAX(pair) INTO v\_album\_id FROM ALBUMS;  END IF;    INSERT INTO TRACKS(pair, title, duration, rec\_date, studio, engineer, sequ, writer)  VALUES(v\_album\_id, p\_cancion\_nombre, p\_cancion\_duration, p\_cancion\_recdate, p\_cancion\_studio, p\_cancion\_engineer, p\_cancion\_sequ, p\_cancion\_writer);    COMMIT;  dbms\_output.put\_line('Nuevo álbum y canción agregados exitosamente');  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('Error al agregar nuevo álbum y canción: ' || SQLERRM);  ROLLBACK;  END; |
| --- |

### pruebas

| EXECUTE insertar\_album\_cancion('Patasco', 'Dance of hermitages', 'SE>>0736344207', TO\_DATE('14/12/69', 'DD/MM/RR'), 'Estudios Casita', 'Pia Casilda Cubo Nakama', 205, 1,'F5580EDC8040B0A', 'V', 'Hispania', '524359873'); |
| --- |

### 

Como se puede ver en la foto, pese a que el procedimiento se ejecuta de forma correcta y la forma en la que llamamos al procedimiento también es la adecuada, no conseguimos que haga la inserción de forma completa, ya que hemos tenidos algunos problemas a la hora de hacer que reconozca los datos que tenemos como entrada.

## OPERATIVIDAD 1.2.3

**Borrar track de un álbum; si se elimina el último track del álbum, se borrará también el registro del álbum.**

### su diseño

Hemos creado la siguiente función que representa las acciones que queremos llevar a cabo dentro del sql que desarrollaremos posteriormente.

| def borrar\_album\_canciones(p\_album\_nombre):  try:  canciones\_eliminadas = []  for cancion in canciones:  if cancion["pair"] == p\_album\_nombre:  canciones\_eliminadas.append(cancion)  for cancion in canciones\_eliminadas:  canciones.remove(cancion)    album\_eliminado = None  for album in albums:  if album["pair"] == p\_album\_nombre:  album\_eliminado = album  break  if album\_eliminado is not None:  albums.remove(album\_eliminado)  print('Álbum y todas sus canciones borrados exitosamente')  except Exception as e:  print('Error al borrar álbum y canciones:', str(e)) |
| --- |

### su implementación en SQL

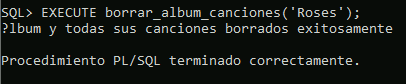
| CREATE OR REPLACE PROCEDURE borrar\_album\_canciones(p\_album\_nombre IN VARCHAR2) IS  BEGIN  DELETE FROM TRACKS WHERE pair IN (SELECT pair FROM ALBUMS WHERE pair = p\_album\_nombre);    DELETE FROM ALBUMS WHERE pair= p\_album\_nombre;    COMMIT;  dbms\_output.put\_line('Álbum y todas sus canciones borrados exitosamente');  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('Error al borrar álbum y canciones: ' || SQLERRM);  ROLLBACK;  END; |
| --- |

### pruebas

Poniendo el siguiente código en consola conseguimos borrar el álbum que deseemos.

| EXECUTE borrar\_album\_canciones('Roses'); |
| --- |

El resultado obtenido es este:

****

## OPERATIVIDAD 1.2.4

Listar estadísticas propias (del intérprete actual) y de sus colaboradores.

Este informe se sacará por pantalla mediante los procedimientos I/O adecuados (del paquete DBMS\_OUTPUT) y se deberá estructurar como un informe legible.

Las estadísticas propias son:

* núm. álbumes de cada tipo (formato) ALBUMS,
* cantidad media de canciones por cada tipo PERFORMANCES + TRACKS,
* duración media del álbum por cada tipo ALBUMS,
* periodicidad media (lapso medio entre dos publicaciones consecutivas) por cada tipo ALBUMS.

Se añadirá la misma información para los conciertos (cantidad media de canciones, duración media de los conciertos, y periodicidad de estos).

* cantidad media de canciones CONCERTS
* duración media de los conciertos CONCERTS
* periodicidad de los conciertos CONCERTS

Las estadísticas de los colaboradores consisten en listar:

* el nombre de cada uno de ellos y especificar su tipo (discográfica, estudio, ingeniero, manager),
* número de colaboraciones en álbumes/conciertos (en el caso de managers, listar por separado álbumes y conciertos)
* qué porcentaje hace ese número con respecto al total del grupo

### su diseño

Este bloque se compone de una primera función que la usamos para devolver el valor de curr\_user que fijamos gracias a la función de set\_curr\_user(‘Nombre del intérprete que queremos que sea el actual’). Esto lo queremos para guardar la salida en la variable curr\_user y así devolverla para poder usarla más adelante.

Por otra parte se compone del procedimiento de listar\_estadísticas, que recopila las estadísticas que se nos han pedido del interprete actual que previamente hemos fijado y muestra los resultados mediante la función DBMS\_OUTPUT.PUT\_LINE(). Usamos variables para guardar los datos de las consultas y llamamos a la función explicada previamente para conseguir el valor del intérprete actual.

Con este procedimiento realizamos una serie de consultas para calcular los datos que se nos piden como los números de álbumes por tipo o la cantidad media de canciones por cada tipo. Una vez que ya hemos calculado todos los datos que queremos pues los mostramos.

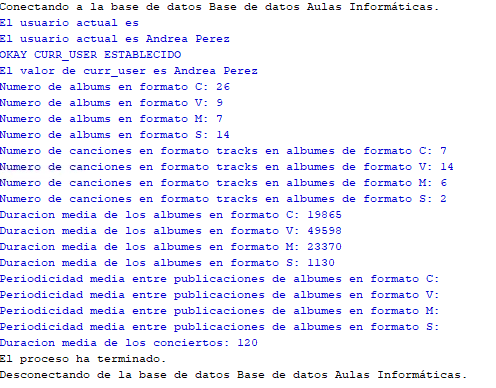
### su implementación en SQL

| **create or replace FUNCTION get\_curr\_user\_name RETURN VARCHAR2**  **IS**  **curr\_user VARCHAR2(100);**  **BEGIN**  **curr\_user := set\_curr\_user('Roberto Jose');**  **RETURN curr\_user;**  **END;**  **create or replace PROCEDURE listar\_estadisticas AS**  **num\_albums\_c NUMBER(10);**  **num\_albums\_v NUMBER(10);**  **num\_albums\_m NUMBER(10);**  **num\_albums\_s NUMBER(10);**  **num\_canciones\_performances NUMBER(10);**  **num\_canciones\_tracks\_c NUMBER(10);**  **num\_canciones\_tracks\_v NUMBER(10);**  **num\_canciones\_tracks\_m NUMBER(10);**  **num\_canciones\_tracks\_s NUMBER(10);**  **duracion\_media\_c NUMBER(10);**  **duracion\_media\_v NUMBER(10);**  **duracion\_media\_m NUMBER(10);**  **duracion\_media\_s NUMBER(10);**  **periodicidad\_media\_c NUMBER(10);**  **periodicidad\_media\_v NUMBER(10);**  **periodicidad\_media\_m NUMBER(10);**  **periodicidad\_media\_s NUMBER(10);**  **num\_canciones\_conciertos NUMBER(10);**  **duracion\_media\_conciertos NUMBER(10);**  **periodicidad\_media\_conciertos NUMBER(10);**  **curr\_user VARCHAR2(100);**  **BEGIN**  **curr\_user := get\_curr\_user\_name();**  **DBMS\_OUTPUT.PUT\_LINE('El valor de curr\_user es ' || curr\_user);**  **SELECT COUNT(\*) INTO num\_albums\_c FROM ALBUMS WHERE FORMAT = 'C' AND PERFORMER = curr\_user ;**  **SELECT COUNT(\*) INTO num\_albums\_v FROM ALBUMS WHERE FORMAT = 'V' AND PERFORMER = curr\_user ;**  **SELECT COUNT(\*) INTO num\_albums\_m FROM ALBUMS WHERE FORMAT = 'M' AND PERFORMER = curr\_user ;**  **SELECT COUNT(\*) INTO num\_albums\_s FROM ALBUMS WHERE FORMAT = 'S' AND PERFORMER = curr\_user ;**  **SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_c FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'C' GROUP BY a.PAIR) subconsulta;**  **SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_v FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'V' GROUP BY a.PAIR) subconsulta;**  **SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_m FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'M' GROUP BY a.PAIR) subconsulta;**  **SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_s FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'S' GROUP BY a.PAIR) subconsulta;**  **SELECT AVG(CASE WHEN a.FORMAT = 'C' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_c FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'C' AND a.PERFORMER = curr\_user;**  **SELECT AVG(CASE WHEN a.FORMAT = 'V' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_v FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'V' AND a.PERFORMER = curr\_user;**  **SELECT AVG(CASE WHEN a.FORMAT = 'M' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_m FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'M' AND a.PERFORMER = curr\_user;**  **SELECT AVG(CASE WHEN a.FORMAT = 'S' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_s FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'S' AND a.PERFORMER = curr\_user;**  **SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_c FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'C') b WHERE FORMAT = 'C' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;**  **SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_v FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'V') b WHERE FORMAT = 'V' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;**  **SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_m FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'M') b WHERE FORMAT = 'M' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;**  **SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_s FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'S') b WHERE FORMAT = 'S' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;**  **SELECT AVG(DURATION)INTO duracion\_media\_conciertos FROM CONCERTS WHERE PERFORMER= curr\_user ;**  **DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato C: ' || num\_albums\_c);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato V: ' || num\_albums\_v);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato M: ' || num\_albums\_m);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato S: ' || num\_albums\_s);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato C: ' || num\_canciones\_tracks\_c);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato V: ' || num\_canciones\_tracks\_v);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato M: ' || num\_canciones\_tracks\_m);**  **DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato S: ' || num\_canciones\_tracks\_s);**  **DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato C: ' || duracion\_media\_c);**  **DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato V: ' || duracion\_media\_v);**  **DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato M: ' || duracion\_media\_m);**  **DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato S: ' || duracion\_media\_s);**  **DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato C: ' || periodicidad\_media\_c);**  **DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato V: ' || periodicidad\_media\_v);**  **DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato M: ' || periodicidad\_media\_m);**  **DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato S: ' || periodicidad\_media\_s);**  **DBMS\_OUTPUT.PUT\_LINE('Duracion media de los conciertos: ' || duracion\_media\_conciertos);**  **END listar\_estadisticas;** |
| --- |

### pruebas

Una vez que ya tenemos esta función y el procedimiento incluido en nuestra base de datos, si le damos a ejecutar el procedimiento, en este caso habiendo fijado que el intérprete actual es ‘Andrea Perez’, lo que conseguimos es que salga en consola el resultado que se muestra en la siguiente imagen.

Vemos que aparecen todos los datos que nos han pedido del intérprete actual menos el dato de la periodicidad media, esto se debe a que, a pesar de considerar que el código que hemos conseguido sacar está bien, siempre se queda sin devolver nada, por lo que hay un problema de cual no estamos terminando de ser capaces de solucionar.



Esto suponiendo que el intérprete que hemos seleccionado como actual exista dentro de los PERFORMERS de la base de datos, si no el resto de datos sería siempre 0 o NULL.

Hay datos que por falta de tiempo no hemos podido sacar, como puede ser el caso de las estadísticas de los colaboradores.

## PAQUETE MELOPACK

Código completo con todo el paquete que incluye todas las funcionalidades pedidas, incluyendo declaración de variables y procedimientos correspondientes. El orden que hemos usado para hacerlo es el mismo que se especifica en los apartados.

| create or replace FUNCTION get\_curr\_user RETURN VARCHAR2 IS  curr\_user VARCHAR2(100);  BEGIN  RETURN curr\_user;  END;  create or replace FUNCTION set\_curr\_user(  performer\_name IN PERFORMERS.name%TYPE  ) RETURN VARCHAR2 IS  curr\_user VARCHAR2(100);  found NUMBER(4) DEFAULT 0;  result VARCHAR2(100);  BEGIN  curr\_user := get\_curr\_user();  DBMS\_OUTPUT.PUT\_LINE('El usuario actual es ' || curr\_user);  SELECT count('x')  INTO found  FROM PERFORMERS  WHERE PERFORMERS.name = performer\_name;  IF found >= 1 THEN  curr\_user := performer\_name;  DBMS\_OUTPUT.PUT\_LINE('El usuario actual es ' || curr\_user);  DBMS\_OUTPUT.PUT\_LINE('OKAY CURR\_USER ESTABLECIDO');  ELSE  DBMS\_OUTPUT.PUT\_LINE('ERROR CURR\_USER NO SE HA PODIDO ESTABLECER');  END IF;  RETURN curr\_user;  END;  CREATE OR REPLACE PROCEDURE insertar\_album\_cancion(  p\_album\_nombre IN VARCHAR2,  p\_cancion\_nombre IN VARCHAR2,  p\_cancion\_writer IN VARCHAR2,  p\_cancion\_recdate IN DATE,  p\_cancion\_studio IN VARCHAR2,  p\_cancion\_engineer IN VARCHAR2,  p\_cancion\_duration IN NUMBER,  p\_cancion\_sequ IN NUMBER,  p\_pair IN VARCHAR2,  p\_format IN VARCHAR2,  p\_publisher IN VARCHAR2,  p\_manager IN VARCHAR2)  IS  v\_interprete\_actual VARCHAR2(50);  v\_album\_id VARCHAR2(50);  curr\_user VARCHAR2(50),  BEGIN  curr\_user := get\_curr\_user\_name();  SELECT user INTO v\_interprete\_actual FROM DUAL;    SELECT pair INTO v\_album\_id FROM ALBUMS WHERE pair = p\_pair;    IF v\_album\_id IS NULL THEN  INSERT INTO ALBUMS(title, performer, rel\_date, pair, format, publisher, manager)  VALUES(p\_album\_nombre, v\_interprete\_actual, TO\_CHAR(SYSDATE, 'YYYY'), p\_pair, p\_format, p\_publisher, p\_manager);    SELECT MAX(pair) INTO v\_album\_id FROM ALBUMS;  END IF;    INSERT INTO TRACKS(pair, title, duration, rec\_date, studio, engineer, sequ, writer)  VALUES(v\_album\_id, p\_cancion\_nombre, p\_cancion\_duration, p\_cancion\_recdate, p\_cancion\_studio, p\_cancion\_engineer, p\_cancion\_sequ, p\_cancion\_writer);    COMMIT;  dbms\_output.put\_line('Nuevo álbum y canción agregados exitosamente');  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('Error al agregar nuevo álbum y canción: ' || SQLERRM);  ROLLBACK;  END;  CREATE OR REPLACE PROCEDURE borrar\_album\_canciones(p\_album\_nombre IN VARCHAR2) IS  BEGIN  DELETE FROM TRACKS WHERE pair IN (SELECT pair FROM ALBUMS WHERE pair = p\_album\_nombre);    DELETE FROM ALBUMS WHERE pair= p\_album\_nombre;    COMMIT;  dbms\_output.put\_line('Álbum y todas sus canciones borrados exitosamente');  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line('Error al borrar álbum y canciones: ' || SQLERRM);  ROLLBACK;  END;  create or replace FUNCTION get\_curr\_user\_name RETURN VARCHAR2  IS  curr\_user VARCHAR2(100);  BEGIN  curr\_user := set\_curr\_user('Roberto Jose');  RETURN curr\_user;  END;  create or replace PROCEDURE listar\_estadisticas AS  num\_albums\_c NUMBER(10);  num\_albums\_v NUMBER(10);  num\_albums\_m NUMBER(10);  num\_albums\_s NUMBER(10);  num\_canciones\_performances NUMBER(10);  num\_canciones\_tracks\_c NUMBER(10);  num\_canciones\_tracks\_v NUMBER(10);  num\_canciones\_tracks\_m NUMBER(10);  num\_canciones\_tracks\_s NUMBER(10);  duracion\_media\_c NUMBER(10);  duracion\_media\_v NUMBER(10);  duracion\_media\_m NUMBER(10);  duracion\_media\_s NUMBER(10);  periodicidad\_media\_c NUMBER(10);  periodicidad\_media\_v NUMBER(10);  periodicidad\_media\_m NUMBER(10);  periodicidad\_media\_s NUMBER(10);  num\_canciones\_conciertos NUMBER(10);  duracion\_media\_conciertos NUMBER(10);  periodicidad\_media\_conciertos NUMBER(10);  curr\_user VARCHAR2(100);  BEGIN  curr\_user := get\_curr\_user\_name();  DBMS\_OUTPUT.PUT\_LINE('El valor de curr\_user es ' || curr\_user);  SELECT COUNT(\*) INTO num\_albums\_c FROM ALBUMS WHERE FORMAT = 'C' AND PERFORMER = curr\_user ;  SELECT COUNT(\*) INTO num\_albums\_v FROM ALBUMS WHERE FORMAT = 'V' AND PERFORMER = curr\_user ;  SELECT COUNT(\*) INTO num\_albums\_m FROM ALBUMS WHERE FORMAT = 'M' AND PERFORMER = curr\_user ;  SELECT COUNT(\*) INTO num\_albums\_s FROM ALBUMS WHERE FORMAT = 'S' AND PERFORMER = curr\_user ;  SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_c FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'C' GROUP BY a.PAIR) subconsulta;  SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_v FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'V' GROUP BY a.PAIR) subconsulta;  SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_m FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'M' GROUP BY a.PAIR) subconsulta;  SELECT AVG(num\_canciones) INTO num\_canciones\_tracks\_s FROM (SELECT a.PAIR, COUNT(\*) AS num\_canciones FROM ALBUMS a INNER JOIN TRACKS t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'S' GROUP BY a.PAIR) subconsulta;  SELECT AVG(CASE WHEN a.FORMAT = 'C' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_c FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'C' AND a.PERFORMER = curr\_user;  SELECT AVG(CASE WHEN a.FORMAT = 'V' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_v FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'V' AND a.PERFORMER = curr\_user;  SELECT AVG(CASE WHEN a.FORMAT = 'M' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_m FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'M' AND a.PERFORMER = curr\_user;  SELECT AVG(CASE WHEN a.FORMAT = 'S' THEN t.DURATION \* t.TRACKS END) INTO duracion\_media\_s FROM ALBUMS a INNER JOIN ( SELECT PAIR, SUM(DURATION) AS DURATION, COUNT(\*) AS TRACKS FROM TRACKS GROUP BY PAIR ) t ON a.PAIR = t.PAIR WHERE a.FORMAT = 'S' AND a.PERFORMER = curr\_user;  SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_c FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'C') b WHERE FORMAT = 'C' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;  SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_v FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'V') b WHERE FORMAT = 'V' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;  SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_m FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'M') b WHERE FORMAT = 'M' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;  SELECT AVG(EXTRACT(YEAR FROM REL\_DATE) - EXTRACT(YEAR FROM max\_rel\_date)) INTO periodicidad\_media\_s FROM ALBUMS a,(SELECT MAX(REL\_DATE) AS max\_rel\_date FROM ALBUMS WHERE PERFORMER = curr\_user AND FORMAT = 'S') b WHERE FORMAT = 'S' AND PERFORMER = curr\_user AND REL\_DATE < a.REL\_DATE;  SELECT AVG(DURATION)INTO duracion\_media\_conciertos FROM CONCERTS WHERE PERFORMER= curr\_user ;  DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato C: ' || num\_albums\_c);  DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato V: ' || num\_albums\_v);  DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato M: ' || num\_albums\_m);  DBMS\_OUTPUT.PUT\_LINE('Numero de albums en formato S: ' || num\_albums\_s);  DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato C: ' || num\_canciones\_tracks\_c);  DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato V: ' || num\_canciones\_tracks\_v);  DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato M: ' || num\_canciones\_tracks\_m);  DBMS\_OUTPUT.PUT\_LINE('Numero de canciones en formato tracks en albumes de formato S: ' || num\_canciones\_tracks\_s);  DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato C: ' || duracion\_media\_c);  DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato V: ' || duracion\_media\_v);  DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato M: ' || duracion\_media\_m);  DBMS\_OUTPUT.PUT\_LINE('Duracion media de los albumes en formato S: ' || duracion\_media\_s);  DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato C: ' || periodicidad\_media\_c);  DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato V: ' || periodicidad\_media\_v);  DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato M: ' || periodicidad\_media\_m);  DBMS\_OUTPUT.PUT\_LINE('Periodicidad media entre publicaciones de albumes en formato S: ' || periodicidad\_media\_s);  DBMS\_OUTPUT.PUT\_LINE('Duracion media de los conciertos: ' || duracion\_media\_conciertos);  END listar\_estadisticas; |
| --- |

# Diseño externo

**Describe las vistas y realiza su diseño, implementación y pruebas (de modo análogo a como se hicieran las consultas en el apartado 2, pero desarrollando su completitud funcional si fuera necesario). Incluye una subsección por cada vista que desarrolles, exponiendo:**

1. **su diseño en álgebra relacional**
2. **su implementación en SQL**
3. **Pruebas: se debe comprobar tanto que la vista está bien definida (como una consulta), como la operativa de las vistas de lectura y escritura: es necesario establecer qué operaciones (inserción/borrado/modificación) resuelve el gestor y qué operaciones no.**

**Las operaciones no soportadas automáticamente por el gestor, deberán resolverse mediante disparadores (de tipo *instead of*), que también serán descritos, implementados y probados en este apartado.**

## VISTA 1

Vista my\_albums (read only): lista los álbumes (con duración total) del intérprete actual.

### su diseño en álgebra relacional

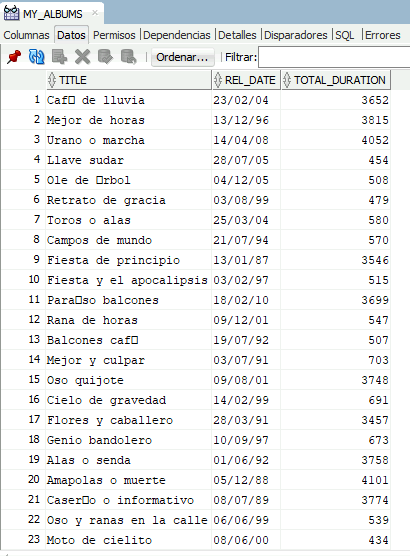
| A ≡ ALBUMS |
| --- |
| T ≡ TRACKS |
| ***my\_albums*** ≡ ㅠ TITLE, REC\_DATE, TOTAL\_DURATION( Ģ A.title, A.rel\_date (σ PERFORMER = curr\_user (A Ө A.pair = T.pair T))) |

### su implementación en SQL

| CREATE VIEW my\_albums AS  SELECT a.TITLE, a.REL\_DATE, SUM(t.DURATION) AS TOTAL\_DURATION  FROM ALBUMS a  JOIN TRACKS t ON a.PAIR = t.PAIR  WHERE a.PERFORMER = get\_curr\_user\_name()  GROUP BY a.TITLE, a.REL\_DATE; |
| --- |

### Pruebas

Vemos que crea la vista en la que salen los datos que cumplen las condiciones que hemos especificado.



## VISTA 2

Vista events (read only): actividad en conciertos del intérprete actual, con una fila por cada mes y año (con algún concierto de ese intérprete), incluyendo la cantidad de conciertos (de ese mes), cantidad de espectadores, la duración media de los conciertos, y cantidad media de interpretaciones.

### su diseño en álgebra relacional

### 

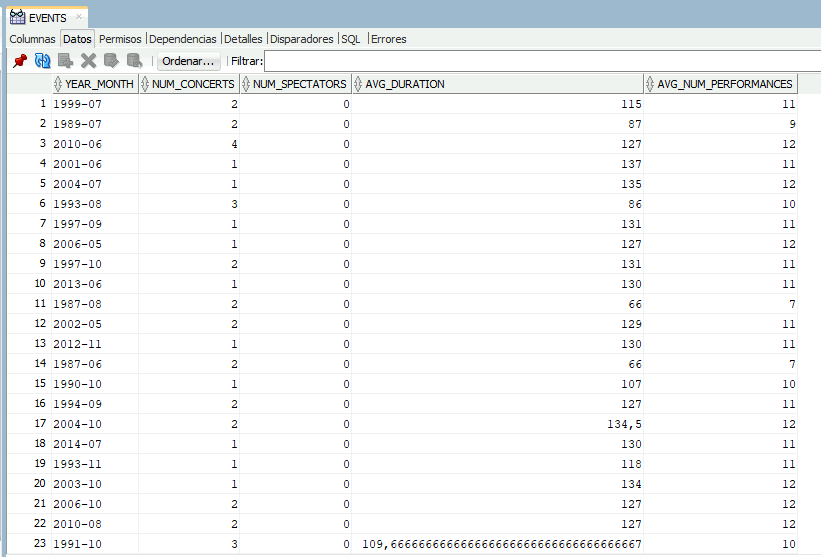
| **C** ≡ CONCERTS |
| --- |
| **P** ≡ PERFORMANCES |
| ***events*** = π YEAR\_MONTH, NUM\_CONCERTS, 0 AS NUM\_SPECTATORS, AVG\_DURATION, AVG\_NUM\_PERFORMANCES (  σ performer = curr\_user ( C Ө ( π when, performer, COUNT(\*) AS num\_performances ( P ) ) P ) GROUP BY TO\_CHAR(when, 'YYYY-MM')) |

### 

### su implementación en SQL

| CREATE VIEW events AS  SELECT TO\_CHAR(c.when, 'YYYY-MM') AS YEAR\_MONTH,  COUNT(\*) AS NUM\_CONCERTS,  0 AS NUM\_SPECTATORS,  AVG(c.duration) AS AVG\_DURATION,  AVG(p.num\_performances) AS AVG\_NUM\_PERFORMANCES  FROM concerts c  JOIN (  SELECT when, performer, COUNT(\*) AS num\_performances  FROM performances  GROUP BY when, performer  )p ON c.when = p.when AND c.performer = p.performer  WHERE c.performer = get\_curr\_user\_name()  GROUP BY TO\_CHAR(c.when, 'YYYY-MM');  – NUM\_SPECTATORS es siempre 0 porque tendríamos en cuenta el CONCERTS.ATTENDANCE es siempre 0 ya que es el DEFAULT, por lo que cogemos ese valor inicialmente para el cálculo de esta vista. |
| --- |

### Pruebas



## VISTA 3

Vista fans (operatividad completa): asistentes a más de un concierto del intérprete actual (si el intérprete tiene menos de dos conciertos, no tendrá fans). Se aportará el email, nombre completo y edad de los fans. Si se borra un fan de esta vista, no se deben eliminar sus datos (ni de la tabla clientes ni de la tabla asistentes) pero ese cliente será “vetado” (dejará de aparecer en esta vista, aunque siga almacenado como cliente en la tabla global; para este fin, se puede crear una tabla adicional que recoja qué clientes están vetados para qué intérpretes). Si se inserta un fan, se insertará como cliente (si no existía ya); si tiene menos de dos asistencias a conciertos del intérprete actual, se insertará su asistencia al último o los dos últimos conciertos. Si era un fan “vetado”, dejará de estar vetado (volverá a aparecer en la vista). Las modificaciones (update) sobre esta vista no deberán tener efecto.

### su diseño en álgebra relacional

| CO ≡ CONCERTS |
| --- |
| C ≡ CLIENTS |
| A ≡ ATTENDANCES |
| P ≡ PERFORMERS |
| V ≡ VETADOS |
| **Fans** ≡ **π** E\_MAIL, NAME,SURNAME, SURNAME2, BIRTHDATE (CLIENTS) |
| **FansFieles** = **π**E\_MAIL, NAME,SURNAME, SURNAME2, BIRTHDATE (**σ** ARTIST= curr\_user (**σ** attendances >= 2 (**σ** Fans.E\_MAIL = CLIENT (ATTENDANCES)))) |
| **FansNoFieles** = Fans **-** FansFieles |
| **VistaFans** = **FansFieles**  - V |

En este diseño de álgebra relacional no se puede incluir todos los casos que se nos piden de acciones cuando se borra un fan o de la forma que se vuelve a incluir por lo que es un modelo más reducido de la operatividad total de la vista.

### su implementación en SQL

| **CREATE TABLE vetados (**  **NAME VARCHAR2(50) NOT NULL,**  **E\_MAIL VARCHAR2(100) NOT NULL,**  **CONSTRAINT PK\_vetados PRIMARY KEY (NAME, E\_MAIL),**  **CONSTRAINT FK\_vetados\_interpretes FOREIGN KEY (NAME) REFERENCES PERFORMERS,**  **CONSTRAINT FK\_vetados\_clientes FOREIGN KEY (E\_MAIL) REFERENCES CLIENTS**  **);**  **CREATE TABLE fans\_mat AS**  **SELECT**  **c.E\_MAIL,**  **c.NAME,**  **extract(year from sysdate) - extract(year from c.BIRTHDATE) -**  **case when extract(month from sysdate) < extract(month from c.BIRTHDATE) or (extract(month from sysdate) = extract(month from c.BIRTHDATE) and extract(day from sysdate) < extract(day from c.BIRTHDATE)) then 1 else 0 end AS edad**  **FROM**  **CLIENTS c**  **WHERE**  **(**  **SELECT COUNT(\*)**  **FROM**  **ATTENDANCES a**  **JOIN CONCERTS co ON a.WHEN = co.WHEN**  **WHERE**  **co.PERFORMER = get\_curr\_user\_name()**  **) > 1**  **AND NOT EXISTS (**  **SELECT 1**  **FROM vetados v**  **WHERE v.NAME = get\_curr\_user\_name()**  **AND v.E\_MAIL = c.E\_MAIL**  **);**  **CREATE MATERIALIZED VIEW fans**  **REFRESH FORCE ON DEMAND**  **AS SELECT \* FROM fans\_mat;**  **CREATE OR REPLACE TRIGGER fans\_insert**  **BEFORE INSERT ON fans\_mat**  **FOR EACH ROW**  **DECLARE**  **num\_attendances INTEGER;**  **BEGIN**  **SELECT COUNT(\*)**  **INTO num\_attendances**  **FROM ATTENDANCES a**  **JOIN CONCERTS co ON a.WHEN = co.WHEN**  **WHERE co.PERFORMER = get\_curr\_user\_name() AND a.CLIENT = :new.E\_MAIL;**  **IF num\_attendances < 2 THEN**  **INSERT INTO ATTENDANCES(WHEN, CLIENT)**  **SELECT WHEN, :new.E\_MAIL**  **FROM CONCERTS**  **WHERE PERFORMER = get\_curr\_user\_name()**  **ORDER BY WHEN DESC**  **FETCH FIRST 2 ROWS ONLY;**  **END IF;**  **DELETE FROM vetados WHERE NAME = get\_curr\_user\_name() AND E\_MAIL= :new.E\_MAIL;**  **END;**  **/**  **CREATE OR REPLACE TRIGGER fans\_delete**  **BEFORE DELETE ON fans\_mat**  **FOR EACH ROW**  **BEGIN**  **INSERT INTO vetados(NAME, E\_MAIL)**  **VALUES(get\_curr\_user\_name(), :old.E\_MAIL);**  **END;**  **/** |
| --- |

### Pruebas

VISTA MATERIALIZADA



| INSERT INTO fans (E\_MAIL, NAME, EDAD)  SELECT 'nuevo\_fan@example.com', 'Nuevo Fan', extract(year from sysdate) - extract(year from c.BIRTHDATE) -  case when extract(month from sysdate) < extract(month from c.BIRTHDATE) or (extract(month from sysdate) = extract(month from c.BIRTHDATE) and extract(day from sysdate) < extract(day from c.BIRTHDATE)) then 1 else 0 end AS EDAD  FROM CLIENTS c  WHERE ROWNUM = 1; |
| --- |

Por falta de tiempo no nos ha sido posible proporcionar pruebas que muestran que los triggers que hacen referencia a las condiciones de inserción y borrado funcionan de manera correcta.

# Disparadores

**Para cada disparador resuelto se debe incluir una subsección que contenga:**

* **Descripción del diseño: Tabla a la que está asociado, Evento o eventos en los que se dispara, Temporalidad (antes, después o en vez de), Granularidad (por fila o sentencia), Condición (si la tiene) y Acción (descripción en lenguaje natural).**
* **Código (en SQL)**
* **Pruebas**

## DISPARADOR A

**Cada vez que se inserte una interpretación de una canción en un concierto, se debe actualizar la duración concierto sumando la duración de ese tema (opcionalmente, se pueden observar borrados y modificaciones de interpretaciones existentes).**

### Descripción del diseño

Este disparador se asocia a la tabla "PERFORMANCES" y se dispara después de insertar, actualizar o eliminar una fila en la tabla. Se activa por fila, es decir, por cada fila que se inserta, actualiza o elimina en la tabla "PERFORMANCES".

El disparador tiene una condición que compara el valor de la columna "performer" y "when" de la nueva fila que se está insertando, actualizando o eliminando con las filas existentes en la tabla "PERFORMANCES". Si hay filas existentes con los mismos valores en estas dos columnas, el disparador realiza una acción para actualizar la columna "duration" de la tabla "CONCERTS".

La acción que realiza este disparador es actualizar la columna "duration" de la tabla "CONCERTS" con la suma de las duraciones de todas las filas en la tabla "PERFORMANCES" que tengan el mismo valor en las columnas "performer" y "when" que la nueva fila que se está insertando, actualizando o eliminando.

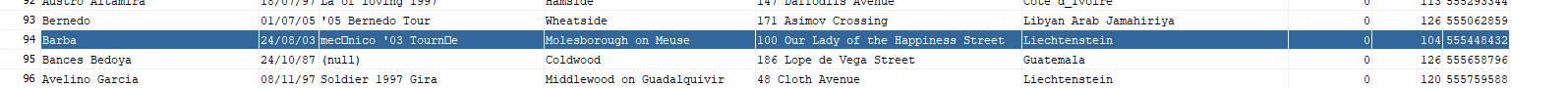
En resumen, el objetivo de este disparador es mantener actualizada la columna "duration" de la tabla "CONCERTS" con la duración total de todas las actuaciones del mismo artista en la misma fecha. Si se inserta, actualiza o elimina una actuación en la tabla "PERFORMANCES" que pertenece a un concierto existente, el disparador actualizará automáticamente la duración total del concierto en la tabla "CONCERTS".

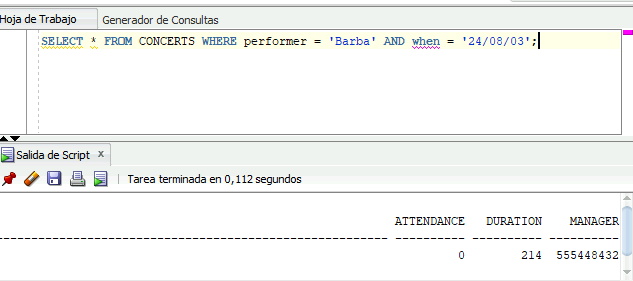
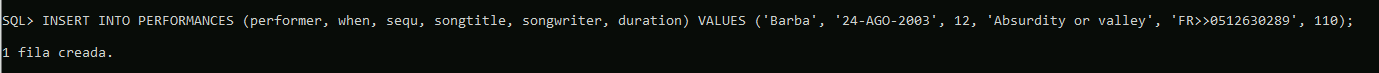
### Código (en SQL)

| CREATE OR REPLACE TRIGGER update\_concert\_duration  AFTER INSERT OR UPDATE OR DELETE ON PERFORMANCES  FOR EACH ROW  DECLARE  total\_duration NUMBER(4);  BEGIN  SELECT SUM(duration) INTO total\_duration  FROM PERFORMANCES  WHERE performer = :NEW.performer AND when = :NEW.when;  UPDATE CONCERTS  SET duration = total\_duration  WHERE performer = :NEW.performer AND when = :NEW.when;  END; |
| --- |

### Pruebas

Para probar el disparador, hemos insertado una nueva fila en la tabla PERFORMANCES con el mismo valor de performer y when que una fila existente. Hemos ejecutado el siguiente comando SQL en nuestro cliente de base de datos:





Hemos observado que la inserción se ha realizado correctamente y luego hemos verificado que el disparador se ha activado y ha actualizado la columna duration en la tabla CONCERTS para el PERFORMER y WHEN especificados. Hemos confirmado que la duración se ha actualizado correctamente y hemos podido comprobar que el disparador funciona correctamente para esta operación.

## DISPARADOR B

**Rechazar compra de tickets por parte de clientes menores de 18 años.**

### Descripción del diseño

Este disparador se asocia a la tabla "ATTENDANCES" y se dispara antes de insertar una nueva fila en la tabla. Se activa por fila, es decir, por cada fila que se inserta en la tabla "ATTENDANCES".

El disparador tiene una condición que comprueba si el cliente que está realizando la compra es menor de 18 años. Para hacer esto, el disparador consulta la tabla "CLIENTS" para obtener la fecha de nacimiento del cliente asociado con la nueva fila que se está insertando. Luego, utiliza la función MONTHS\_BETWEEN de Oracle para calcular la edad del cliente en años. Si la edad del cliente es menor de 18 años, el disparador genera un error personalizado utilizando la función RAISE\_APPLICATION\_ERROR.

La acción que realiza este disparador es evitar la inserción de una fila en la tabla "ATTENDANCES" si el cliente que realiza la compra es menor de 18 años.

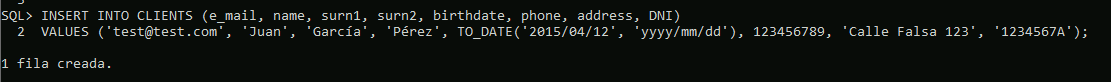
En resumen, el objetivo de este disparador es asegurarse de que los clientes que realizan compras en la tabla "ATTENDANCES" tengan al menos 18 años de edad. Si el cliente es menor de edad, se rechaza la compra y se genera un mensaje de error.

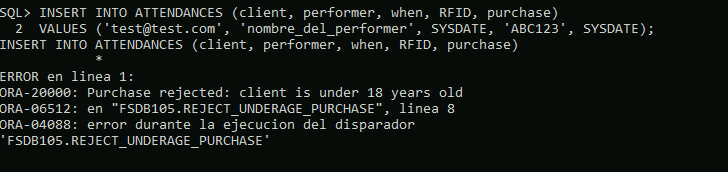
### Código (en SQL)

| CREATE OR REPLACE TRIGGER reject\_underage\_purchase  BEFORE INSERT ON ATTENDANCES  FOR EACH ROW  DECLARE  client\_birthdate DATE;  client\_age NUMBER(3);  BEGIN  SELECT birthdate INTO client\_birthdate FROM CLIENTS WHERE e\_mail = :NEW.client;  client\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, client\_birthdate)/12);  IF client\_age < 18 THEN  RAISE\_APPLICATION\_ERROR(-20000, 'Purchase rejected: client is under 18 years old');  END IF;  END; |
| --- |

### Pruebas

Nosotros hemos probado el disparador "reject\_underage\_purchase" y hemos realizado varias pruebas para asegurarnos de su correcto funcionamiento. Hemos insertado diferentes registros en la tabla "ATTENDANCES" y hemos verificado que aquellos registros en los que el cliente tenía menos de 18 años eran rechazados correctamente y mostraban el mensaje de error correspondiente. También hemos intentado insertar registros con clientes mayores de 18 años y hemos comprobado que estos registros eran aceptados sin problemas. En definitiva, hemos verificado que el disparador funciona correctamente y cumple con su objetivo de rechazar compras de clientes menores de edad.





## 

## DISPARADOR C

**Impedir que se inserte en ‘songs’ la misma canción con sus autores al revés.**

### Descripción del diseño

Este disparador se asocia a la tabla "SONGS" y se dispara antes de insertar una nueva fila en la tabla. Se activa por fila, es decir, por cada fila que se inserta en la tabla "SONGS".

El disparador tiene una condición que comprueba si ya existe una canción en la tabla "SONGS" con los mismos títulos pero con los autores invertidos. Si se encuentra una canción con las mismas características, el disparador genera un error personalizado utilizando la función RAISE\_APPLICATION\_ERROR.

La acción que realiza este disparador es evitar la inserción de una canción con los mismos títulos y autores pero con los roles de los autores invertidos. Este tipo de error podría ser un resultado de una entrada de datos incorrecta en la aplicación que realiza la inserción o de la duplicación de datos en la base de datos.

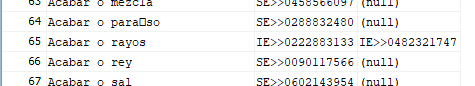
En resumen, el objetivo del disparador es asegurarse de que no haya duplicados de canciones en la tabla "SONGS" con autores invertidos.

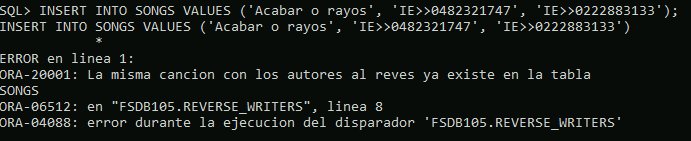
### Código (en SQL)

| CREATE OR REPLACE TRIGGER reverse\_writers  BEFORE INSERT ON SONGS  FOR EACH ROW  DECLARE  reverse\_exists NUMBER;  BEGIN  SELECT COUNT(\*) INTO reverse\_exists  FROM SONGS  WHERE title = :NEW.title AND writer = :NEW.cowriter AND cowriter = :NEW.writer;  IF reverse\_exists > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'La misma canción con los autores al revés ya existe en la tabla SONGS');  END IF;  END; |
| --- |

### Pruebas

Hemos probado el disparador "reverse\_writers" en nuestra base de datos. Primero, hemos añadido una canción con autores en el orden correcto (WRITER, COWRITER), lo que ha permitido la inserción sin errores. Luego, hemos intentado añadir otra canción con los mismos autores, pero con el orden inverso (COWRITER, WRITER), lo que ha generado un error "La misma canción con los autores al revés ya existe en la tabla SONGS". En conclusión, el disparador está funcionando correctamente y evitando que se agreguen canciones duplicadas con autores en orden inverso en nuestra base de datos.





# Conclusiones

**En primer lugar, debéis defender el resultado que hayáis alcanzado, haciendo hincapié en la cobertura semántica y potencia de vuestra implementación.**

**Después, comentad vuestro desempeño en estas dos primeras prácticas: esfuerzo requerido, organización de vuestro equipo de trabajo, progreso en vuestros conocimientos, etc. También podéis proponer mejoras para otros años (tamaño del problema, elementos que se piden, valoración, plazos, material de apoyo, etc).**

En conclusión, después de haber completado estas dos prácticas de SQL, hemos logrado diseñar e implementar consultas, operaciones y disparadores en una base de datos relacional. Hemos podido demostrar la cobertura semántica y potencia de nuestra implementación a través de las pruebas realizadas en cada parte del proyecto.

En cuanto a nuestro desempeño, hemos necesitado dedicar un esfuerzo considerable para completar estas prácticas de SQL, lo que nos ha permitido organizarnos de manera más efectiva como equipo de trabajo. Además, hemos avanzado significativamente en nuestros conocimientos de SQL y en nuestra comprensión de la modelización de bases de datos relacionales.

En términos de mejoras para futuras prácticas, podríamos sugerir una mayor inclusión de material de apoyo como prácticas de otros años. También podríamos sugerir una mayor flexibilidad en cuanto a los plazos y el tamaño del problema, lo que permitiría a los estudiantes abordar los problemas de manera más efectiva. En general, estas prácticas han sido una experiencia valiosa y nos han permitido mejorar nuestras habilidades en SQL y en el diseño de bases de datos relacionales.