

# EVALUACIÓN FINAL MODULO 5

---

Autor : Jorge Córdova



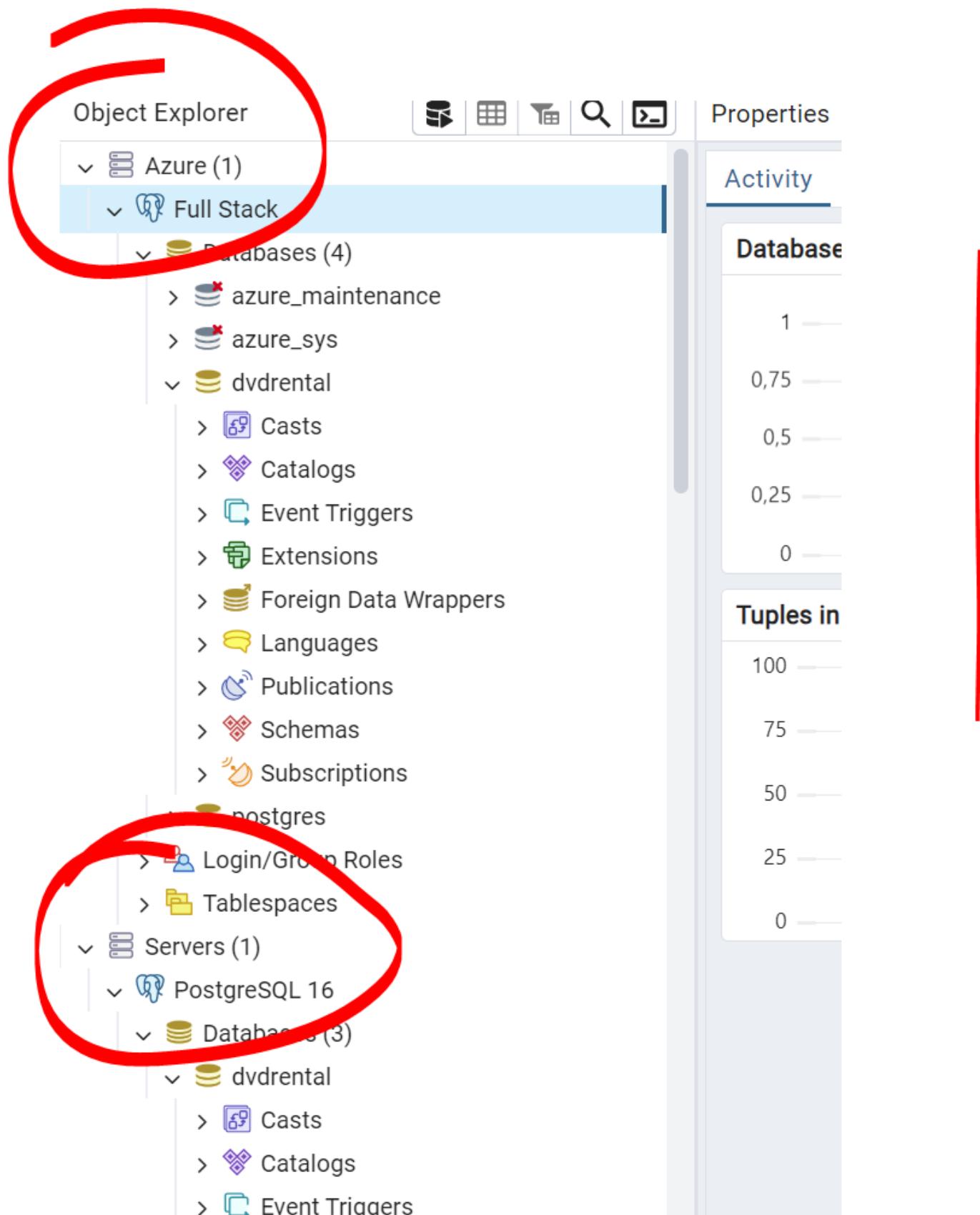
## Introducción

El presente trabajo es parte de los requisitos de evaluación al término del módulo 5, del bootcamp Full Stack Python, relativo a conocimientos sobre sql, usando el motor de base de datos PostgreSQL v. 16.4.

## Análisis del requerimiento

Para los ejercicios propuestos se ha utilizado la base de datos de ejemplo, provista por pgSQL, llamada "dvdrental", relacionada con el arrendamiento de películas.

Como se observa en la imagen, se montó la base en dos servidores:



- En Azure: Se aprovechó el ejercicio para montar una instancia en un servidor en la nube.
- En Localhost: Se montaron dos instancias ("dvrental" y "dvrentalTest").
- En MongoDB: mediante la exportación de las tablas en formato CSV, se montó la misma base sólo con fines académicos.

The screenshot shows the Compass MongoDB interface. On the left, there's a sidebar with 'My Queries' and 'CONNECTIONS (1)'. Under 'CONNECTIONS', there's a tree view with 'pruebas\_fullstack' expanded, showing 'admin', 'config', and 'dvdrental'. 'dvdrental' is selected and expanded, showing sub-collections: actor, address, category, city, country, customer, film, film\_actor, film\_category, inventory, and language. The main area displays four collection statistics cards: 'actor', 'address', 'category', and 'city'. Each card provides storage size, document count, average document size, index count, and total index size.

Collection	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
actor	24.58 kB	200	120.00 B	1	20.48 kB
address	57.34 kB	603	177.00 B	1	24.58 kB
category	20.48 kB	16	93.00 B	1	20.48 kB
city	20.48 kB	400	107.00 B	1	20.48 kB

Las tareas requeridas son:

1. Cargar una base de datos desde un archivo de respaldo: Para el cumplimiento de la actividad, se importó el archivo proporcionado ".tar" a PostgreSQL. Nota: Los ejercicios fueron desarrollados en la instancia "dvdrentalTest", para que, en la segunda instancia ("dvdrental"), se pudiesen replicar todos los pasos desde 0, en la misma secuencia y así obtener los mismos resultados.
2. Escribir consultas SQL: Se entiende que se deben crear consultas básicas de tipo CRUD, y también, consultas más complejas para obtener información específica, por ejemplo, aplicando agrupaciones.
3. Comprender el modelo relacional: Para poder armar las distintas consultas y operaciones sobre la base de datos, se requiere analizar y entender la estructura de la base de datos "dvdrental", las relaciones entre sus tablas, restricciones y otros elementos que forman parte del esquema original.
4. Documentar la base de datos: Se solicita también, crear un diccionario de datos que describa las tablas y columnas.
5. Crear un backup: Finalmente, para poder auditar las operaciones solicitadas, se debe generar un respaldo con los cambios producidos en la BBDD para su comprobación.

## Restaurar la BBDD "DVDRENTAL"

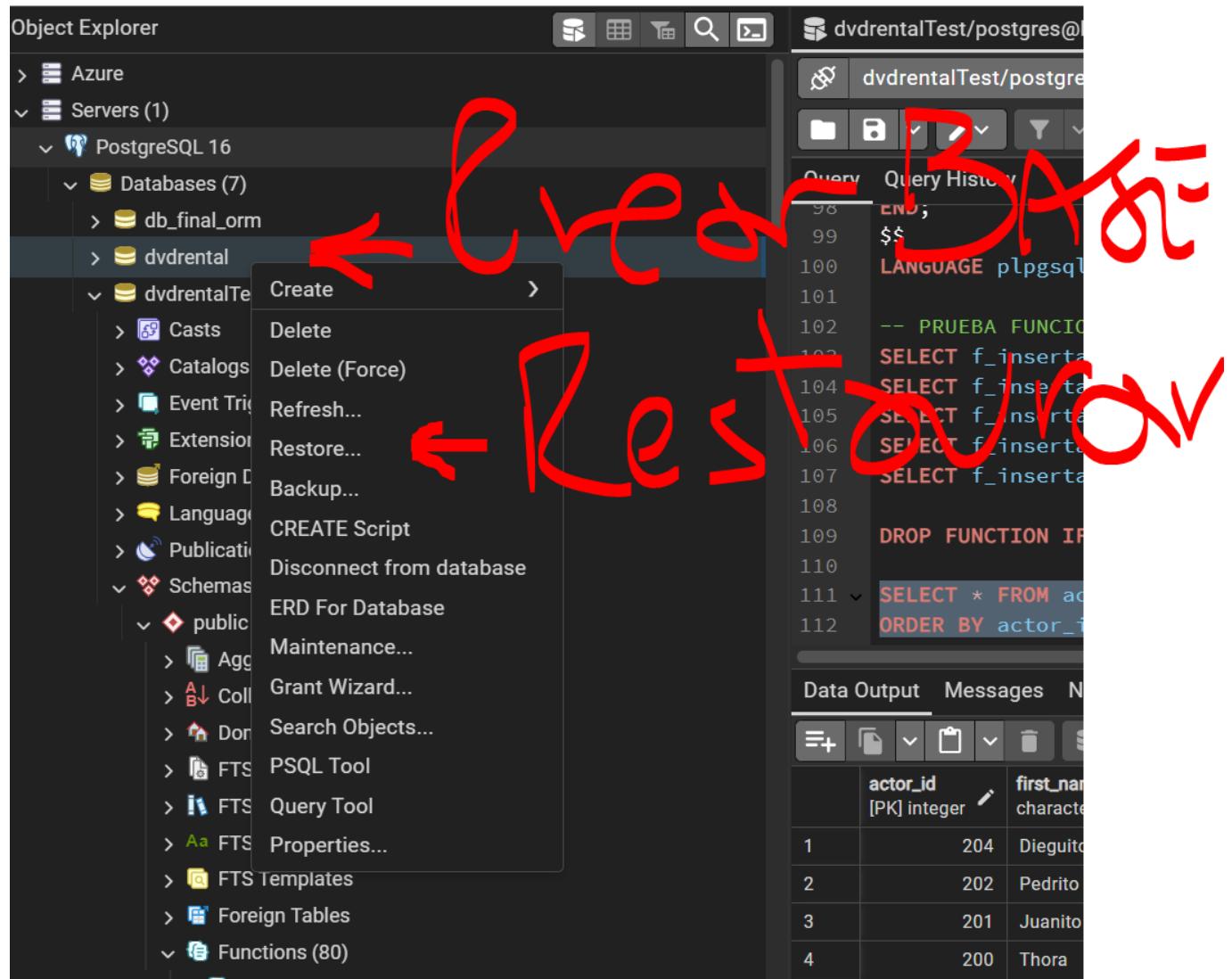
La descripción no especifica cómo, sin embargo, para restaurar la base se puede hacer a través de línea de comandos, o utilizando la interfaz gráfica que, con pocas acciones permite montar el esquema y los datos en la nueva base de datos de destino.

Se realizó el procedimiento de restauración a través de la descompresión del archivo ".tar" de la bbdd.

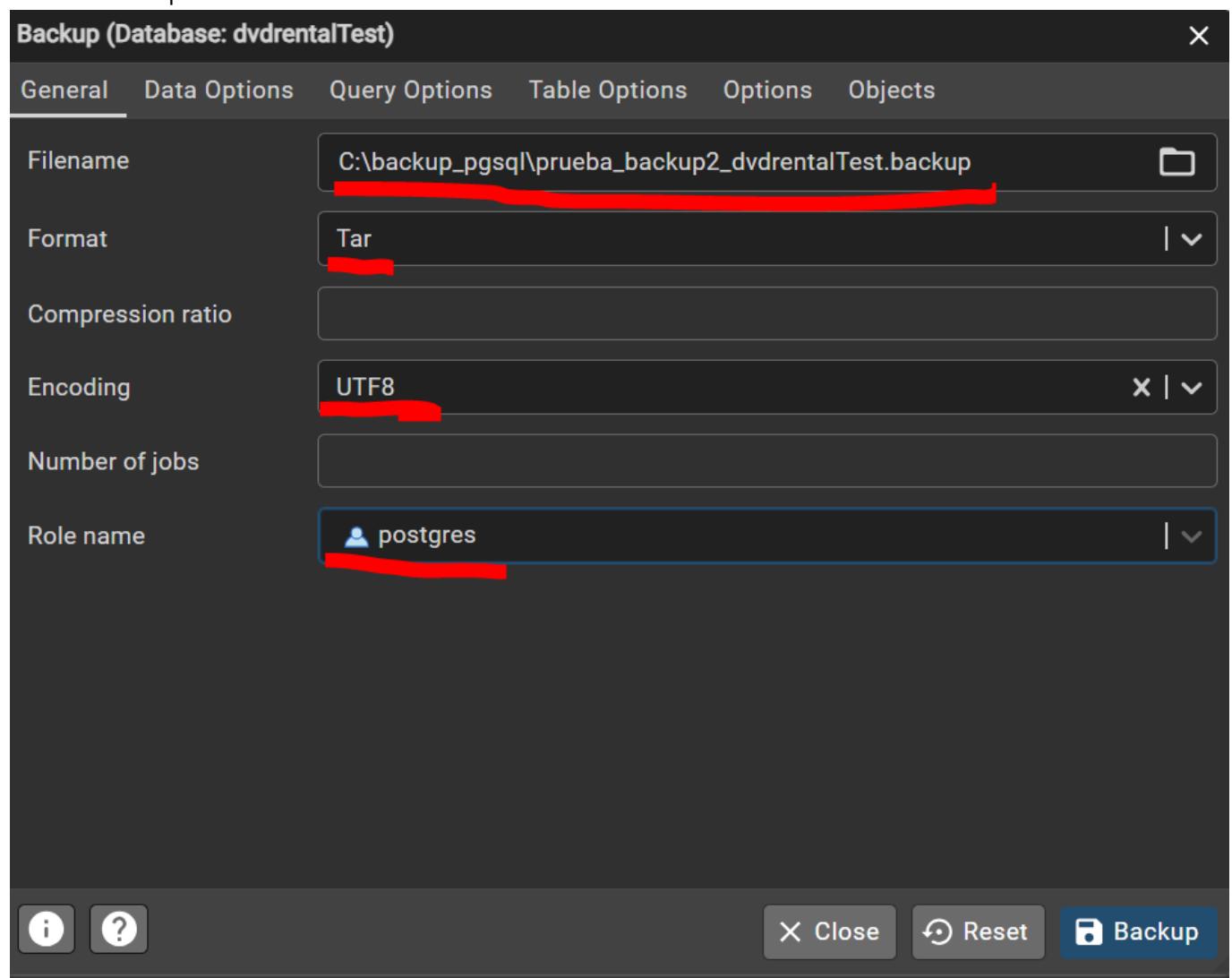
Se realizó el mismo procedimiento tanto para instanciar la BBDD en Azure, como en localhost.

Para probar todos los métodos, también se probaron por consola, usando psql.

## Restaurar BBDD desde PgAdmin



Indicar la ruta para el archivo de restauración ".tar"



## Antes de empezar

- **Eliminar o reemplazar una función**

Cada vez que una función se modifica, debe ser actualizada pero, para ello, si la función ya existía, requiere ser eliminada previamente. Esta es una tarea habitual.

Esta sentencia se puede escribir "a mano" o, se puede usar la que provee el propio pgAdmin, haciendo click derecho sobre el objeto, luego "Scripts/CREATE/Create Script". Esta acción abrirá una pestaña mostrando el código y, en la parte superior, se hallará la sentencia para dropear la función, sin cometer errores de sintaxis.

## PROCEDIMIENTOS GENERALES DE INSERCIÓN

### 1. OPERACIONES DE INSERCIÓN PARA LA ENTIDAD CUSTOMER

- **Insertar un cliente de forma manual ("create" CUSTOMER)**

Para insertar un nuevo registro en la tabla customer se requiere hacer dos operaciones secuenciales.

Primero, crear la dirección, en su defecto se generará un error de FK, si se intenta, insertar un customer, sin su dirección.

Segundo, realizar una inserción simple, pero lo vamos a resolver con una función que haga ambas cosas.

- **Datos de ejemplo:**

Para generar datos aleatorios se creo una función en Python usando la librería FAKER, todos los datos generados para prueba se generan y guardan en la carpeta:

\data\_pruebas\_[fecha].csv

Las tuplas generadas para este caso son:

```
```sql
('Jorge', 'Cordova', 'jorge.cordova@mymail.com', 'Calle Siempreviva
123', 'Santiago', 1, 1, '12345', '9555-12121');
('Jota', 'Juillerat', 'jota.juillerat@elmail.com', 'Calle Elm 456',
'Santiago', 1, 1, '67890', '9444-21212', 1); ('Alejandro', 'Juille',
'alejuille@mumail.com', 'Calle Test 789', 'Santiago', 1, 1, '13579',
'9333-31313', 1);
('Ale', 'Cordova', 'ale_cordova@gmail.com', 'Calle Nueva 2354',
'Santiago', 1, 1, '75000', '9666-43434', 1);
('Cindy Test', 'Gonzalez Test', 'cynthiatesest@testmail.net', '753
Carlson Courts, MA 61349', 'Davidfort', 1, 1, '50600', '736.643.4357',
1);
('Teresa Garcia Test', 'Test Brady', 'TereGarcia@example.net', '068
Samuel Islands Apt. 711, IN 45377', 'Joneston', 1, 1, '17589',
'(905)654-0273', 0);
```

```

Luego de chequear varias veces, logré establecer los campos que son necesarios para conservar la integridad y respetar los constraints (por ejemplo, no nulo o requerido).

La lógica es simple, se capturan todos los datos necesarios, luego se insertan en el orden correcto, primero address y luego customer, así se garantiza que se cumplan la integridad y las restricciones.

- **Script 'DROP' función que insertar cliente (CUSTOMER)**

Este es el último script para dropear la función que inserta un nuevo customer en la tabla.

```
```sql
DROP FUNCTION IF EXISTS f_insertar_cliente(character varying,
character varying, character varying, character varying, character
varying, integer, integer, character varying, character varying,
integer);
```

```

- **FUNCTION: f\_insertar\_cliente()**

```

```sql
CREATE OR REPLACE FUNCTION f_insertar_cliente( _first_name
VARCHAR(45), _last_name VARCHAR(45), _email VARCHAR(50), _address
VARCHAR(50), _district VARCHAR(20), _city_id INT, _store_id INT,
_postal_code VARCHAR(10), _phone VARCHAR(20), _active INT )
RETURNS VOID AS $$

DECLARE _address_id INT;

BEGIN -- Primero insertamos la dirección, si no existe.
    INSERT INTO address (address, district, city_id, postal_code,
    phone)
        VALUES (_address, _district, _city_id, _postal_code, _phone)
    RETURNING address_id
        INTO _address_id; -- Luego, podemos insertar un nuevo cliente
    sin errores
        INSERT INTO customer (first_name, last_name, email, address_id,
    store_id, active)
            VALUES (_first_name, _last_name, _email, _address_id,
    _store_id, _active);
        END;
    $$

LANGUAGE plpgsql;
```

```

- **Datos de ejemplo insertados**

La primera versión funcional del insert, generó que el campo 'active' del registro **customer\_id Nr.602**, quedase vacío (null), sin embargo, se aprovechará este error para aplicar el procedimiento de actualización o update en la sección siguiente

```

```sql
SELECT f_insertar_cliente('Jorge', 'Cordova',
'jorge.cordova@mymail.com', 'Calle Siempreviva 123', 'Santiago', 1, 1,
'12345', '9555-12121');

SELECT f_insertar_cliente('Jota', 'Juillerat',
'jota.juillerat@elmail.com', 'Calle Elm 456', 'Santiago', 1, 1,
'67890', '9444-21212', 1);

SELECT f_insertar_cliente('Alejandro', 'Juille',
'alejuille@mumail.com', 'Calle Test 789', 'Santiago', 1, 1, '13579',
'9333-31313', 1);

SELECT f_insertar_cliente('Ale', 'Cordova', 'ale_cordova@gmail.com',
'Calle Nueva 2354', 'Santiago', 1, 1, '75000', '9666-43434', 1);

SELECT f_insertar_cliente('Cindy Test', 'Gonzalez Test',
'cynthiatest@testmail.net', '753 Carlson Courts, MA 61349',
'Davidfort', 1, 1, '50600', '736.643.4357', 1);

SELECT f_insertar_cliente('Teresa Garcia Test', 'Test Brady',
```

```

```
'TereGarcia@example.net', '068 Samuel Islands Apt. 711, IN 45377',
'Joneston', 1, 1, '17589', '(905)654-0273', 0);
```

![1731419931693](image/evalFinalM5_Jorge_Cordova/1731419931693.png)

![1731419993663](image/evalFinalM5_Jorge_Cordova/1731419993663.png)
```

- **Listar los registros insertados**

```
```sql
SELECT * FROM customer WHERE last_name = 'Cordova' SELECT * FROM
customer WHERE last_name = 'Juille' SELECT * FROM customer WHERE
last_name = 'Juillerat' SELECT * FROM customer WHERE last_name =
'Juillerat' OR last_name = 'Juille' OR last_name = 'Cordova'
```

```

The screenshot shows a database interface with a toolbar at the top labeled 'Data Output', 'Messages', and 'Notifications'. Below the toolbar is a SQL query editor window containing the following code:

```
customer_id | store_id | first_name | last_name | email | address_id | activebool | create_date | last_update
602 | 1 | Jorge | Cordova | jorge.cordova@mymail... | 614 | true | 2024-09-25 | 2024-09-25
603 | 1 | Jota | Juillerat | jota.juillerat@elmail.com | 615 | true | 2024-09-25 | 2024-09-25
604 | 1 | Alejandro | Juille | alejuille@mumail.com | 616 | true | 2024-09-25 | 2024-09-25
```

The table has columns: customer\_id, store\_id, first\_name, last\_name, email, address\_id, activebool, create\_date, and last\_update. The data shows three new entries added to the table.

Para comprobar que los registros se han agregado, también podemos listar los últimos registros de la tabla customer.

```
```sql
SELECT * FROM customer ORDER BY customer_id DESC LIMIT 5
```

```

The screenshot shows a database interface with a toolbar at the top labeled 'Data Output', 'Messages', and 'Notifications'. Below the toolbar is a SQL query editor window containing the following code:

```
1 | SELECT * FROM public.customer
2 | ORDER BY customer_id DESC LIMIT 100
3 |
4 | SELECT * FROM customer
5 | ORDER BY customer_id DESC LIMIT 5
```

The table has columns: customer\_id, store\_id, first\_name, last\_name, email, address\_id, activebool, create\_date, and last\_update. The data shows the last five entries from the table.

La comprobación no puede ser completa, sin listar los cambios de la tabla "address". Listamos las address de los últimos 5 registros agregados

```
```sql
SELECT * FROM address ORDER BY address_id DESC LIMIT 5
```

```

| address_id | address                               | address2 | district  | city_id | postal_code | phone         |
|------------|---------------------------------------|----------|-----------|---------|-------------|---------------|
| 1          | Vivaceta 321                          | Recoleta | Recoleta  | 576     | 812345      | 9444-2222     |
| 2          | Recoleta 123                          | Santiago | Santiago  | 300     | 912345      | 9555-5555     |
| 3          | 068 Samuel Islands Apt. 711, IN 45377 | [null]   | Joneston  | 1       | 17589       | (905)654-0273 |
| 4          | 753 Carlson Courts, MA 61349          | [null]   | Davidfort | 1       | 50600       | 736.643.4357  |
| 5          | Calle Siempreviva 123                 | [null]   | Santiago  | 1       | 12345       | 9555-12121    |

Para indagar sobre las restricciones utilicé la siguiente consulta, que extrae los nombres de las restricciones únicas definidas en una tabla, por ejemplo: "customer" o "address"

P.D: No sirvió de mucho, se entendería que no hay restricciones, más allá de not null o FK.

```
```sql
SELECT conname FROM pg_constraint WHERE contype = 'u' AND conrelid =
'customer'::regclass; SELECT conname FROM pg_constraint WHERE contype =
'u' AND conrelid = 'address'::regclass;
```

```

## 2. OPERACIONES DE INSERCIÓN PARA LA ENTIDAD STAFF

- **Insertar un nuevo empleado (STAFF)**

- Consideraciones:

1. Se debe tener cuidado al insertar un nuevo registro porque hace referencia a una tabla externa ("address"), a través de su clave foránea "address\_id".
2. Adicionalmente, la tabla "address", también posee referencia a otras tablas a través de la FK "city\_id", la que a su vez, está vinculada a la FK "country\_id".
3. Se podría generar una función que valide estos aspectos, introduciendo una capa adicional de verificación, por ejemplo, comprobando si la ciudad y el país existen, lanzando una excepción del tipo:

```
```sql
IF _city_id IS NULL AND NOT EXISTS (SELECT 1 FROM country
WHERE country_id = _country_id) THEN RAISE EXCEPTION 'La
ciudad y/o el país no existen.'; END IF;
```

```

Sin embargo, esto detendría el proceso.

4. También se podrían insertar la ciudad y el país, en caso de no existir, agregando a nuestra función una consulta como esta:

```
```sql
IF _city_id IS NULL THEN INSERT INTO city(city, country_id,
last_update) VALUES (_city_name, _country_id, NOW())
RETURNING city_id INTO _city_id; INSERT INTO country(country,
last_update) VALUES WHERE NOT EXISTS (SELECT 1 FROM country
WHERE country_id = _country_id); END IF;
```

```

5. Estas dos aproximaciones al problema de lograr mayor integridad en los datos para no generar datos basura que luego no conecten con nada, tienen pros y contras. Por ejemplo, podría no ser deseado que se cree un país que no existe (ficticio), o que sea un proceso administrativo que requiere aprobación para modificar un mantenedor. También podría ocurrir que el detectar si la ciudad o país no existen, lanzando una excepción, no sea un comportamiento deseado para la aplicación. En ambos casos, el beneficio es una bbdd consistente, íntegra y útil para los usos que se le quiera dar. Para efectos académicos, habiendo entendido el problema, se reducirá la complejidad, omitiendo estos escenarios y creando datos de pruebas con los datos existentes en los mantenedores (países y ciudades ya creados)

- **FUNCTION: f\_insertar\_staff()**

```
```sql
CREATE OR REPLACE FUNCTION f_insertar_staff( _first_name text,
                                             _last_name text,
                                             _address text,
                                             _address2 text,
                                             _district text,
                                             _city_id integer,
                                             _postal_code text,
                                             _phone text,
                                             _email text,
                                             _store_id integer,
                                             _active boolean,
                                             _username text,
```

```

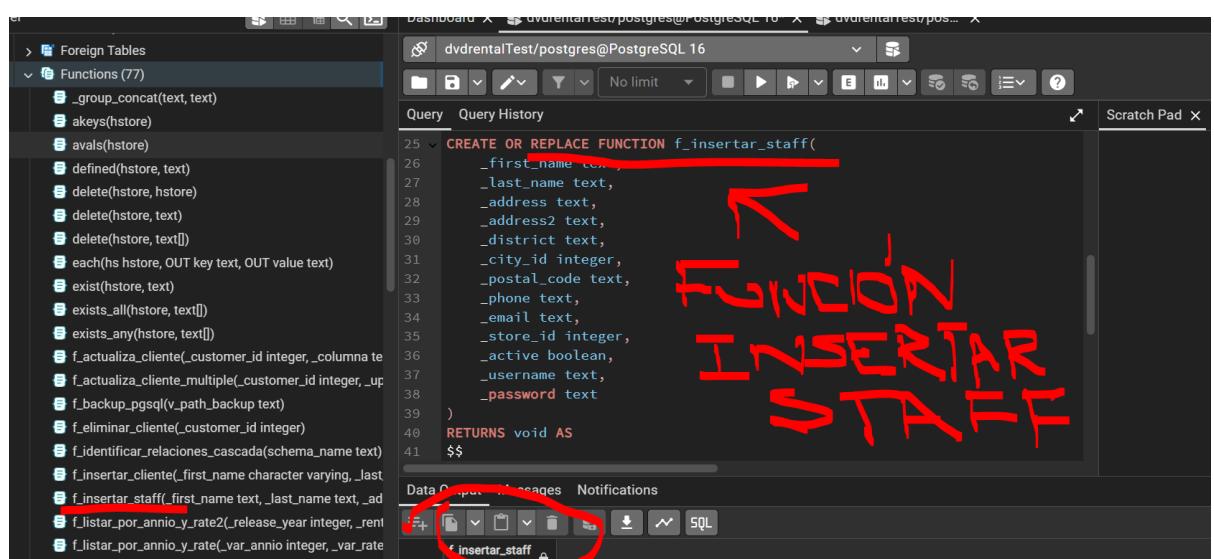
```

        _password text
    )
RETURNS void AS $$

DECLARE v_address_id integer;
BEGIN -- Primero insertamos la dirección para mantener la consistencia
de BBDD -- Para el city_id, usaremos datos existentes en la BBDD
    INSERT INTO address( address, address2, district, city_id,
postal_code, phone, last_update )
    VALUES ( _address, _address2, _district, _city_id, _postal_code,
_address, NOW() ) RETURNING address_id
        INTO v_address_id; -- Insertar el empleado
    INSERT INTO staff( first_name, last_name, address_id, email,
store_id, active, username, password, last_update )
    VALUES ( _first_name, _last_name, v_address_id, _email, _store_id,
_active, _username, _password, NOW() );
END; $$

LANGUAGE plpgsql;
```

```



- **Eliminar la función `f_insertar_staff()`**

```

```sql
DROP FUNCTION IF EXISTS f_eliminar_staff(integer);
```

```

- **Datos de prueba `f_insertar_staff()`**

Referencia) (first\_name, last\_name, address, address2, district, city\_id, postal\_code, phone, email, store\_id, active, username, password)

```

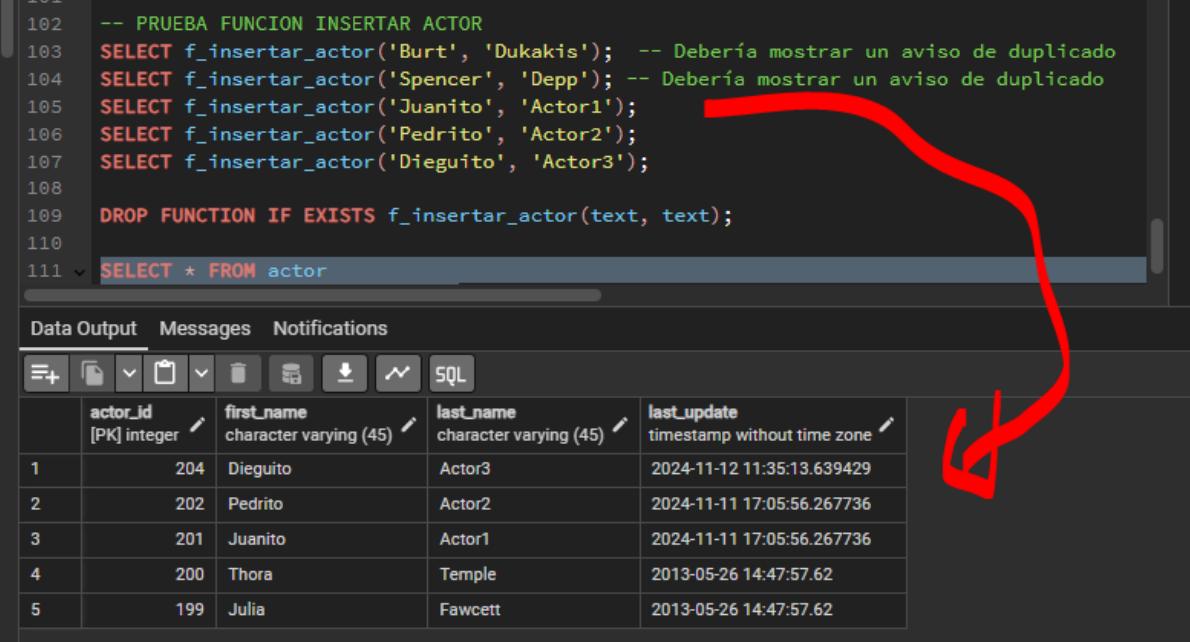
```sql
('Pedrito', 'Staff1', 'Recoleta 123', 'Santiago', 'Santiago', 300,

```

```
'912345', '9555-5555', 'pstaff1e@eval5.com', 2, true, 'Pedrito1',
'pass123')
('Juanito', 'Staff2', 'Vivaceta 321', 'Recoleta', 'Recoleta', 576,
'812345', '9444-2222', 'jstaff2@eval5.com', 1, true, 'Juanito2',
'pass123')
('Dieguito', 'Staff3', 'Apoquindo 3321', 'Las Condes', 'Las Condes',
300, '912345', '8333-5555', 'dstaff3@eval5.com', 2, true, 'Dieguito3',
'pass123')
('Fulanito', 'Staff4', 'Alameda 8321', 'Santiago', 'Santiago', 576,
'812345', '2555-1111', 'fstaff4@eval5.com', 2, true, 'Fulanito4',
'pass123')
('Sutanito', 'Staff5', 'Matta 2321', 'Santiago', 'Santiago', 300,
'912345', '3123-1231', 'sstaff5@eval5.com', 1, true, 'Sutanito5',
'pass123')
```
```

- Ejecución

```
```sql
SELECT f_insertar_staff('Pedrito', 'Staff1', 'Recoleta 123',
'Santiago', 'Santiago', 300, '912345', '9555-5555',
'pstaff1e@eval5.com', 2, true, 'Pedrito1', 'pass123');
SELECT f_insertar_staff('Juanito', 'Staff2', 'Vivaceta 321',
'Recoleta', 'Recoleta', 576, '812345', '9444-2222',
'jstaff2@eval5.com', 1, true, 'Juanito2', 'pass123');
SELECT f_insertar_staff('Dieguito', 'Staff3', 'Apoquindo 3321', 'Las
Condes', 'Las Condes', 300, '912345', '8333-5555', 'dstaff3@eval5.com',
2, true, 'Dieguito3', 'pass123');
SELECT f_insertar_staff('Fulanito', 'Staff4', 'Alameda 8321',
'Santiago', 'Santiago', 576, '812345', '2555-1111',
'fstaff4@eval5.com', 2, true, 'Fulanito4', 'pass123');
SELECT f_insertar_staff('Sutanito', 'Staff5', 'Matta 2321',
'Santiago', 'Santiago', 300, '912345', '3123-1231',
'sstaff5@eval5.com', 1, true, 'Sutanito5', 'pass123');
```
```



```

102 -- PRUEBA FUNCION INSERTAR ACTOR
103 SELECT f_insertar_actor('Burt', 'Dukakis'); -- Debería mostrar un aviso de duplicado
104 SELECT f_insertar_actor('Spencer', 'Depp'); -- Debería mostrar un aviso de duplicado
105 SELECT f_insertar_actor('Juanito', 'Actor1');
106 SELECT f_insertar_actor('Pedrito', 'Actor2');
107 SELECT f_insertar_actor('Dieguito', 'Actor3');
108
109 DROP FUNCTION IF EXISTS f_insertar_actor(text, text);
110
111 SELECT * FROM actor

```

Data Output Messages Notifications

|   | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
|---|--------------------------|--------------------------------------|-------------------------------------|--|
| 1 | 204                      | Dieguito                             | Actor3                              | 2024-11-12 11:35:13.639429                 |
| 2 | 202                      | Pedrito                              | Actor2                              | 2024-11-11 17:05:56.267736                 |
| 3 | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 4 | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |
| 5 | 199                      | Julia                                | Fawcett                             | 2013-05-26 14:47:57.62                     |

### 3. OPERACIONES DE INSERCIÓN PARA LA ENTIDAD ACTOR

- **Insertar un nuevo actor (ACTOR)**

- Consideraciones:

1. No existen restricciones evidentes al momento de crear un nuevo actor, porque esta tabla es referida por la tabla "film", pudiendo un actor, no tener asociadas películas al momento de ser creado.
2. Entendiendo cómo funciona la tabla, vamos a crear una función que inserte nuevos registros en la tabla actor, pero antes de hacer la inserción, verificará si el actor ya existe en la base de datos, evitando así duplicados.

- **FUNCTION: f\_insertar\_actor()**

```

```sql
CREATE OR REPLACE FUNCTION f_insertar_actor(
    _first_name text,
    _last_name text
)
RETURNS void AS
$$
DECLARE
    _actor_id integer;
BEGIN
    -- Verificamos si el actor ya existe
    SELECT actor_id INTO _actor_id
    FROM actor
    WHERE first_name = _first_name AND last_name = _last_name;
    -- Si no existe, insertamos el nuevo actor
    IF _actor_id IS NULL THEN
        INSERT INTO actor(
            first_name, last_name, last_update
        )

```

```

)
VALUES (
    _first_name, _last_name, NOW()
);
ELSE
    RAISE NOTICE 'El actor % % ya existe.', _first_name, _last_name;
END IF;
END;
$$
LANGUAGE plpgsql;
```

```

## CREAR FUNCIÓN

```

CREATE OR REPLACE FUNCTION f_insertar_actor(
    _first_name text,
    _last_name text
)
RETURNS void AS
$$
DECLARE
    _actor_id integer;
BEGIN
    -- Verificamos si el actor ya existe
    SELECT actor_id INTO _actor_id
    FROM actor
    WHERE first_name = _first_name AND last_name = _last_name;
    -- Si no existe, insertamos el nuevo actor
    IF _actor_id IS NULL THEN
        INSERT INTO public.actor(
            first_name, last_name, last_update
        )
        VALUES (
            _first_name, _last_name, NOW()
        );
    END IF;
END;
$$
LANGUAGE plpgsql;
```

```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 145 msec.

- Eliminar la función `f_insertar_actor()`

```

```sql
DROP FUNCTION IF EXISTS f_insertar_actor(text, text);
```

```

- Datos de prueba: `f_insertar_actor()`

```

```sql
('Burt', 'Dukakis'); -- El actor ya existe en la tabla, lanza el
aviso de duplicado
('Spencer', 'Depp'); -- El actor ya existe en la tabla, lanza el
aviso de duplicado
('Juanito', 'Actor1'); -- Registro nuevo, aparece al final de la
tabla ACTOR
('Pedrito', 'Actor2'); -- Registro nuevo, aparece al final de la
tabla ACTOR
```

```

```
('Dieguito', 'Actor3'); -- Registro nuevo, aparece al final de la
tabla ACTOR
```

```

- Ejecución

```
```sql
SELECT f_insertar_actor('Burt', 'Dukakis'); -- Debe mostrar un aviso
de duplicado
SELECT f_insertar_actor('Spencer', 'Depp'); -- Debe mostrar un aviso
de duplicado
SELECT f_insertar_actor('Juanito', 'Actor1');
SELECT f_insertar_actor('Pedrito', 'Actor2');
SELECT f_insertar_actor('Dieguito', 'Actor3');
```

```

No inserta datos duplicados

```
Query  Query History
90      first_name, last_name, last_update
91      )
92      VALUES (
93          _first_name, _last_name, NOW()
94      );
95      ELSE
96          RAISE NOTICE 'El actor %% ya existe.', _first_name, _last_name;
97      END IF;
98  END;
99  $$;
100 LANGUAGE plpgsql;
101 -- PRUEBA FUNCION INSERTAR ACTOR
102 SELECT f_insertar_actor('Burt', 'Dukakis'); -- Debería mostrar un aviso de duplicado
Data Output  Messages  Notifications
f_insertar_actor  void
1

```

Hace validación a través de evaluar los argumentos

```
Query  Query History
94      );
95      ELSE
96          RAISE NOTICE 'El actor %% ya existe.', _first_name, _last_name;
97      END IF;
98  END;
99  $$;
100 LANGUAGE plpgsql;
101 -- PRUEBA FUNCION INSERTAR ACTOR
102 SELECT f_insertar_actor('Burt', 'Dukakis'); -- Debería mostrar un aviso de duplicado
103 SELECT f_insertar_actor('Spencer', 'Depp'); -- Debería mostrar un aviso de duplicado
104 SELECT f_insertar_actor('Juanito', 'Actor1');
105 SELECT f_insertar_actor('Pedrito', 'Actor2');
106 SELECT f_insertar_actor('Dieguito', 'Actor3');
Data Output  Messages  Notifications
f_insertar_actor  void
1

```

Sólo inserta los nuevos registros

```
Query History
1 SELECT * FROM public.actor
2 ORDER BY actor_id desc LIMIT 10
3
```

|   | actor_id | first_name | last_name   | last_update                |
|---|----------|------------|-------------|----------------------------|
| 1 | 203      | Dieguito   | Actor3      | 2024-11-11 17:05:56.267736 |
| 2 | 202      | Pedrito    | Actor2      | 2024-11-11 17:05:56.267736 |
| 3 | 201      | Juanito    | Actor1      | 2024-11-11 17:05:56.267736 |
| 4 | 200      | Thora      | Temple      | 2013-05-26 14:47:57.62     |
| 5 | 199      | Julia      | Fawcett     | 2013-05-26 14:47:57.62     |
| 6 | 198      | Mary       | Keitel      | 2013-05-26 14:47:57.62     |
| 7 | 197      | Reese      | West        | 2013-05-26 14:47:57.62     |
| 8 | 196      | Bela       | Walken      | 2013-05-26 14:47:57.62     |
| 9 | 195      | Ivana      | Silverstone | 2013-05-26 14:47:57.62     |

Total rows: 10 of 10    Query complete 00:00:00.087    Ln 1, Col 1

## PROCEDIMIENTOS GENERALES DE ELIMINACIÓN

### 1. OPERACIONES DE ELIMINACIÓN PARA LA ENTIDAD CUSTOMER

- **Eliminar un cliente de forma manual ("DELETE" CUSTOMER)**

Forma simple, eliminamos el cliente con 2 consultas, Una para el customer y la otra para la dirección

Primero eliminamos el customer por su ID de customer. Se eliminará el customer\_id = 603

```
-- Forma simple, eliminamos el cliente con 2 consultas
-- Una para el customer y la otra para la dirección
-- Primero eliminamos el customer por su ID de customer
-- Se eliminará el customer_id = 603
DELETE FROM customer
WHERE customer_id = 603; -- El address_id = 615
-- Segundo, eliminamos el address por su ID de customer
DELETE FROM address
WHERE address_id = 615; -- El address_id = 615
/*
```

|   | customer_id | store_id | first_name              | last_name     | email                             | address_id | activebool | create_date | last_update             |
|---|-------------|----------|-------------------------|---------------|-----------------------------------|------------|------------|-------------|-------------------------|
| 1 | 607         | 1        | Uso de Procedure UPDATE | Test Brady    | TereGarcia@example.net            | 619        | true       | 2024-09-27  | 2024-09-30 18:00:15.214 |
| 2 | 606         | 1        | Test2 uso de fx update  | Gonzalez Test | cindystest2_updated@example.com   | 618        | true       | 2024-09-27  | 2024-09-30 18:14:36.149 |
| 3 | 603         | 1        | Jota                    | Juillerat     | jota.juillerat@email.com          | 615        | true       | 2024-09-25  | 2024-09-25 10:06:18.028 |
| 4 | 602         | 1        | Jorge                   | Cordova       | jorge.cordova@mymail.com          | 614        | true       | 2024-09-25  | 2024-09-25 10:57:26.729 |
| 5 | 599         | 2        | Austin                  | Cintron       | austin.cintron@sakilacustomer.org | 605        | true       | 2006-02-14  | 2013-05-26 14:49:45.736 |

```
```sql
-- Primero eliminamos el customer 603
DELETE FROM customer WHERE customer_id = 603;
```

```
-- Tiene el address_id = 615
-- Segundo, eliminamos el address_id 615
DELETE FROM address WHERE address_id = 615;
```

```

Query History

```
114 -- Forma simple, eliminamos el cliente con 2 consultas
115 -- Una para el customer y la otra para la dirección
116
117 -- Primero eliminamos el customer por su ID de customer
118 -- Se eliminará el customer_id = 603
119
120 DELETE FROM customer
    WHERE customer_id = 603; -- El address_id = 615
121
122
123 -- Segundo, eliminamos el address
124
125 DELETE FROM address
    WHERE address_id = 615;
126
127
```

Data Output

```
DELETE 1

Query returned successfully in 87 msec.
```

# 603

### Registro en customer borrado

Query History

```
114 -- Forma simple, eliminamos el cliente con 2 consultas
115 -- Una para el customer y la otra para la dirección
116
117 -- Primero eliminamos el customer por su ID de customer
118 -- Se eliminará el customer_id = 603
119
120 DELETE FROM customer
    WHERE customer_id = 603; -- El address_id = 615
121
122
123 -- Segundo, eliminamos el address
124
125 DELETE FROM address
    WHERE address_id = 615;
126
127
```

Data Output

| customer_id | store_id | first_name              | last_name     | email                             | address_id | activebool | create_date | last_update             |
|-------------|----------|-------------------------|---------------|-----------------------------------|------------|------------|-------------|-------------------------|
| 1           | 607      | Uso de Procedure UPDATE | Test Brady    | TereGarcia@example.net            | 619        | true       | 2024-09-27  | 2024-09-30 18:00:15.24  |
| 2           | 606      | Test2 uso de fx update  | Gonzalez Test | cindytest2_updated@example.com    | 618        | true       | 2024-09-27  | 2024-09-30 18:14:36.149 |
| 3           | 602      | 1 Jorge                 | Cordova       | jorge.cordova@mymail.com          | 614        | true       | 2024-09-25  | 2024-09-25 10:57:26.725 |
| 4           | 599      | 2 Austin                | Cintron       | austin.cintron@sakilacustomer.org | 605        | true       | 2006-02-14  | 2013-05-26 14:49:45.736 |
| 5           | 598      | 1 Wade                  | Delvalle      | wade.delvalle@sakilacustomer.org  | 604        | true       | 2006-02-14  | 2013-05-26 14:49:45.736 |

Se borrarón  
el #603

### Registro en address borrado

Query History

```
98 SELECT * FROM address
99 ORDER BY address_id DESC LIMIT 5
100
101 -- Para indagar sobre las restricciones utilice la siguiente consulta
102 -- Extrae los nombres de las restricciones únicas definidas en la tabla
103 -- "customer" o "address".
104 -- P.D: No sirvió de mucho, se entendería que no hay restricciones, más
105 -- allá de nat null o FK
```

Data Output

| address_id | address                                   | address2 | district     | city_id | postal_code | phone         | last_update        |
|------------|---|----------|--------------|---------|-------------|---------------|--------------------|
| 1          | 619 068 Samuel Islands Apt. 711, IN 45377 | [null]   | Joneston     | 1       | 17589       | (905)654-0273 | 2024-09-27 15:49:3 |
| 2          | 618 753 Carlson Courts, MA 61349          | [null]   | Davidfort    | 1       | 50600       | 736.643.4357  | 2024-09-27 14:31:1 |
| 3          | 614 Calle Siempreviva 123                 | [null]   | Santiago     | 1       | 12345       | 9555-12121    | 2024-09-25 10:57:2 |
| 4          | 605 1325 Fukuyama Street                  |          | Heilongjiang | 537     | 27107       | 288241215394  | 2006-02-15 09:45:3 |
| 5          | 604 1331 Usak Boulevard                   |          | Vaud         | 296     | 61960       | 145308717464  | 2006-02-15 09:45:3 |

no existe #603

- **Eliminar un cliente usando una función**

Se crea la función "f\_eliminar\_cliente(integer)", que ELIMINA UN CLIENTE DE LA TABLA CUSTOMER.

Primero, se debe verificar si hay "borrado en cascada", en su defecto, se debe manejar el procedimiento (manual o a través de funciones) para no dejar datos inconsistentes o dark data.

La siguiente consulta, fue la primera versión que no borra la dirección, pero se dejó para mostrar como manejar el error.

- **Primera versión de la función (No borra address)**

```
```sql
CREATE OR REPLACE FUNCTION f_eliminar_cliente(_customer_id INT)
RETURNS VOID AS $$ BEGIN DELETE FROM customer
WHERE customer_id = _customer_id;
END;
$$
LANGUAGE plpgsql;
```
```

```

Ejecutamos la función y eliminamos un registro de la tabla customer por su id = 605

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date
1	605	1	Ale	Cordova	ale_cordova@gmail.com	617	true	2024-
2	604	1	Alejandro	Juille	alejuille@mumail.com	616	true	2024-
3	603	1	Jota	Juillerat	jota.juillerat@elmail.com	615	true	2024-
4	602	1	Jorge	Cordova	jorge.cordova@mymail....	614	true	2024-

```
```sql
SELECT f_eliminar_cliente(605)
```
```

```

Como se observa en la BBDD, el script original, sólo eliminaba el customer, pero no su dirección (address), por lo que procedemos a eliminarla manualmente, antes de corregir la función y luego, probaremos nuevamente con la nueva versión.

	address_id [PK] Integer	address character varying (50)	address2 character varying (50)	district character varying (20)	city_id smallint	postal_code character varying (10)	phone character varying (20)	last_update timestamp
1	617	Calle Nueva 2354	[null]	Santiago	1	75000	9666-43434	2024-0
2	616	Calle Test 789	[null]	Santiago	1	13579	9333-31313	2024-0
3	615	Calle Elm 456	[null]	Santiago	1	67890	9444-21212	2024-0
4	614	Calle Siempreviva 123	[null]	Santiago	1	12345	9555-12121	2024-0
5	605	1325 Fukuyama Street		Heilongjiang	537	27107	288241215394	2006-0

```
```sql
DELETE FROM address
WHERE address_id = 617; -- El address_id correspondiente al
customer_id 605
```

```

Eliminamos la función anterior, antes de guardar las modificaciones

```
```sql
DROP FUNCTION IF EXISTS f_eliminar_cliente(integer);
```

```

Habiendo observado el error, procedemos a corregir la función, ahora si elimina el cliente y su dirección. La función recibe un customer\_id, y lo primero que hace es buscar el address\_id por el customer\_id, y almacena el dato en una variable. Luego, procesa la eliminación de la dirección, que debe ser previo al customer y, finalmente, elimina el customer por su customer\_id.

- FUNCTION: **f\_eliminar\_cliente()**, versión corregida

```
```sql
CREATE OR REPLACE FUNCTION f_eliminar_cliente(_customer_id INT)
RETURNS VOID AS $$
DECLARE _address_id INT;
BEGIN -- Primero, se debe obtener el address_id a partir del
customer_id
SELECT address_id
INTO _address_id
FROM customer
WHERE customer_id = _customer_id; -- Segundo, se debe eliminar el
cliente, antes que la dirección, por un constraint (FK)
DELETE FROM customer
WHERE customer_id = _customer_id; -- Finalmente, se puede eliminar
la dirección
DELETE FROM address
WHERE address_id = _address_id;
END;
$$

```

```
LANGUAGE plpgsql;
```

```

- **Ejecutamos la eliminación de un registro CUSTOMER**

Ahora, con la función corregida, eliminaremos el cliente "customer\_id = 604" y su dirección asociada "address\_id = 616"

```

180
181 -- Ejecutar eliminación
182 -- *****
183
184 SELECT f_eliminar_cliente(604); -- Eliminaremos el customer_id(604) y address(616)
185
186 -- Listar los filos con calificación mayor a 4.0 (rental_rate) del año 2006 (release date)
187 SELECT film_id AS "Código de la Película", title AS "Título de la Película", rental_rate AS

```

Data Output Messages Notifications

| f_eliminar_cliente | void |
|--------------------|------|
| 1                  |      |

SQL

O Se ejecutó delete #604

```
```sql
SELECT f_eliminar_cliente(604);
```

```

Como se puede comprobar en la BBDD, el registro no existe en la tabla CUSTOMER y tampoco en la tabla ADDRESS asociada por su address\_id.

```

75 -- Para comprobar que los registros se han agregado, podemos
76 -- listar los últimos registros de la tabla customer
77
78 SELECT * FROM public.customer
79 ORDER BY customer_id DESC LIMIT 5
80

```

Data Output Messages Notifications

| customer_id | store_id | first_name | last_name     | email                             | address_id | activebool | create_date | last_update |
|-------------|----------|------------|---------------|-----------------------------------|------------|------------|-------------|-------------|
| 1           | 606      | Cindy Test | Gonzalez Test | cynthiatest@testmail.net          | 618        | true       | 2024-09-27  | 2024-09     |
| 2           | 603      | Jota       | Juillerat     | jota.juillerat@email.com          | 615        | true       | 2024-09-25  | 2024-09     |
| 3           | 602      | Jorge      | Cordova       | jorge.cordova@mymail.com          | 614        | true       | 2024-09-25  | 2024-09     |
| 4           | 599      | Austin     | Cintron       | austin.cintron@sakilacustomer.org | 605        | true       | 2006-02-14  | 2013-05     |
| 5           | 598      | Wade       | Delvalle      | wade.delvalle@sakilacustomer.org  | 604        | true       | 2006-02-14  | 2013-05     |

Se ha eliminado el #604



```

85  SELECT * FROM public.address
86  ORDER BY address_id DESC LIMIT 5
87
88  -- Para indagar sobre las restricciones utilice este comando
89  -- Extrae los nombres de las restricciones únicas definidas en
90  -- la tabla "customer" o "address".

```

|   | address_id | address                      | address2 | district     | city_id | postal_code | phone        | last_update |
|---|------------|------------------------------|----------|--------------|---------|-------------|--------------|-------------|
| 1 | 618        | 733 Carlson Courts, MA 61349 | [null]   | Davidfort    | 1       | 50600       | 736.643.4357 | 2024-0      |
| 2 | 615        | Calle Elm 456                | [null]   | Santiago     | 1       | 67890       | 9444-21212   | 2024-0      |
| 3 | 614        | Calle Siempreviva 123        | [null]   | Santiago     | 1       | 12345       | 9555-12121   | 2024-0      |
| 4 | 605        | 1325 Fukuyama Street         |          | Heilongjiang | 537     | 27107       | 288241215394 | 2006-0      |

Para verificar, se pueden listar últimos registros de las tablas afectadas.

```

```sql
SELECT * FROM customer ORDER BY customer_id DESC LIMIT 10 -- No se
puede ver el registro 604

SELECT * FROM address ORDER BY address_id DESC LIMIT 10 -- No se
puede ver el registro 616
```

```

## 2. OPERACIONES DE ELIMINACIÓN PARA LA ENTIDAD STAFF

- **Eliminar un empleado ("DELETE" STAFF)**

Forma simple, eliminamos el empleado haciendo una consulta a la tabla "staff" y una consulta a la tabla "address".

Para este ejemplo, de los empleados que creamos antes, usaremos el empleado #7, cuyo address\_id es #624.

```

```sql
-- Primero, eliminamos el registro del empleado su id de empleado

DELETE FROM staff
WHERE staff_id = 7;
```

```

Query Query History

```

1 SELECT * FROM staff
2 ORDER BY staff_id ASC LIMIT 10
3
4 SELECT * FROM address
5 ORDER BY address_id DESC LIMIT 10

```

Data Output Messages Notifications

| staff_id | first_name | last_name | address_id | address                      | address2 | district   | city_id | postal_code | phone         | last_update         |
|----------|------------|-----------|------------|------------------------------|----------|------------|---------|-------------|---------------|---------------------|
| 1        | Mike       | Hillyer   | 3          | Mike.Hillyer@sakilastaff.com |          | Santiago   | 576     | 802345      | 2555-1111     | 2023-11-14 10:23:45 |
| 2        | Jon        | Stephens  | 4          | Jon.Stephens@sakilastaff.com |          | Las Condes | 300     | 912345      | 8333-5555     | 2023-11-14 10:23:45 |
| 3        | Pedrito    | Staff1    | 620        | pstaff1e@eval5.com           |          | Recoleta   | 576     | 812345      | 9444-2222     | 2023-11-14 10:23:45 |
| 4        | Juanito    | Staff2    | 621        | jstaff2f@eval5.com           |          | Santiago   | 300     | 912345      | 9555-5555     | 2023-11-14 10:23:45 |
| 5        | Dieguito   | Staff3    | 622        | dstaff3@eval5.com            |          | Joneston   | 1       | 17589       | (905)654-0273 | 2023-11-14 10:23:45 |
| 6        | Fulanito   | Staff4    | 623        | fstaff4@eval5.com            |          | Davidfort  | 1       | 50600       | 736.643.4357  | 2023-11-14 10:23:45 |

```

```sql
-- Segundo eliminamos la dirección del empleado por su id de
dirección:

DELETE FROM address
WHERE address_id = 624;
```

```

Query Query History

```

1 SELECT * FROM staff
2 ORDER BY staff_id ASC LIMIT 10
3
4 SELECT * FROM address
5 ORDER BY address_id DESC LIMIT 10

```

Data Output Messages Notifications

| address_id | address                               | address2   | district     | city_id | postal_code | phone         |
|------------|---------------------------------------|------------|--------------|---------|-------------|---------------|
| 623        | Alameda 8321                          | Santiago   | Santiago     | 576     | 812345      | 2555-1111     |
| 622        | Apoquindo 3321                        | Las Condes | Las Condes   | 300     | 912345      | 8333-5555     |
| 621        | Vivaceta 321                          | Recoleta   | Recoleta     | 576     | 812345      | 9444-2222     |
| 620        | Recoleta 123                          | Santiago   | Santiago     | 300     | 912345      | 9555-5555     |
| 619        | 068 Samuel Islands Apt. 711, IN 45377 | [null]     | Joneston     | 1       | 17589       | (905)654-0273 |
| 618        | 753 Carlson Courts, MA 61349          | [null]     | Davidfort    | 1       | 50600       | 736.643.4357  |
| 614        | Calle Siempreviva 123                 | [null]     | Santiago     | 1       | 12345       | 9555-12121    |
| 605        | 1325 Fukuyama Street                  |            | Hellongjiang | 537     | 27107       | 288241215394  |
| 604        | 1331 Usak Boulevard                   |            | Vaud         | 296     | 61960       | 145308717464  |
| 603        | 1103 Quilmes Boulevard                |            | Piura        | 503     | 52137       | 644021380889  |

```

```sql
-- Se verifica con las siguientes consultas
SELECT * FROM staff
ORDER BY staff_id ASC LIMIT 10

SELECT * FROM address
ORDER BY address_id DESC LIMIT 10
```

```

Usando una función, eliminaremos el empleado pasando como parámetro el número de su id (staff\_id), la misma función hará la consulta a la tabla "address" y, usando su "address\_id",

eliminará la dirección.

Por dependencia, se debe eliminar primero el empleado, y luego la dirección.

Para el ejemplo, eliminaremos el registro #6 de la tabla staff, cuyo address\_id es el 623.

- **Eliminar la función (DROP)**

```
```sql
DROP FUNCTION IF EXISTS f_eliminar_empleado(integer);
```

```

- **FUNCTION: f\_eliminar\_empleado()**

```
```sql
CREATE OR REPLACE FUNCTION f_eliminar_empleado(_staff_id integer)
RETURNS void AS $$ 
DECLARE _address_id integer;
BEGIN -- Antes de eliminar el empleado, necesitamos guardar el ID de
la dirección
    -- porque de otra forma se pierde y no se puede completar el
segundo delete
    SELECT address_id
    INTO _address_id
    FROM staff
    WHERE staff_id = _staff_id; -- Eliminamos el empleado por su id
    DELETE FROM staff
    WHERE staff_id = _staff_id; -- Ahora sí podemos eliminar la
dirección usando el id guardado IF FOUND THEN DELETE FROM address WHERE
address_id = _address_id;
    END IF;
END;
$$
LANGUAGE plpgsql;
```

```

- **Ejecutamos**

```
```sql
SELECT f_eliminar_empleado(6); SELECT f_eliminar_empleado(5);
```

```

```

30 END;
31 $$ LANGUAGE plpgsql;
32
33
34 SELECT f_eliminar_empleado(6);

```

f\_eliminar\_empleado  
void

1

ELIMINAR #6

```

```sql
-- Se verifica con las siguientes consultas

SELECT * FROM staff
ORDER BY staff_id DESC LIMIT 10

SELECT * FROM address
ORDER BY address_id DESC LIMIT 10

-- Ambos registros fueron correctamente eliminados
```

```

| staff_id | first_name | last_name | address_id | email                         | store_id | active | username  |
|----------|------------|-----------|------------|-------------------------------|----------|--------|-----------|
| 1        | Mike       | Hillyer   | 3          | Mike.Hillyer@sakilastaff.com  | 1        | true   | Mike      |
| 2        | Jon        | Stephens  | 4          | Jon.Stephens@sakilastaff.c... | 2        | true   | Jon       |
| 3        | Pedrito    | Staff1    | 620        | pstaff1@eval5.com             | 2        | true   | Pedrito1  |
| 4        | Juanito    | Staff2    | 621        | jstaff2@eval5.com             | 1        | true   | Juanito2  |
| 5        | Dieguito   | Staff3    | 622        | dstaff3@eval5.com             | 2        | true   | Dieguito3 |

#6 NO EXISTE

```

26 delete(hstore, hstore)
27 delete(hstore, text)
28 delete(hstore, text[])
29 each(hs hstore, OUT key text, OUT value text)
30 exist(hstore, text)
31 exists_all(hstore, text[])
32 exists_any(hstore, text[])
33 f_actualiza_cliente(_customer_id integer, _customer_name text)
34 f_actualiza_cliente_multiple(_customer_ids integer[], _customer_names text[])
35 f_backup_pgsql(v_path_backup text)
36 f_eliminar_cliente(_customer_id integer)
37 f_eliminar_empleado(_staff_id integer)
38 f_identificar_relaciones_cascada(schema_name text)
39 f_insertar_cliente(_first_name character varying(45), _last_name character varying(45), _email character varying(50), _address_id smallint, _city_id smallint, _district character varying(20), _postal_code character varying(10), _phone character varying(20))
40 f_listar_por_anio_y_rate2(release_year integer, var_anno integer)
41 SELECT f_eliminar_empleado(6);

```

| address_id | address                               | address2 | district   | city_id | postal_code | phone         |
|------------|---------------------------------------|----------|------------|---------|-------------|---------------|
| 622        | Apoquindo 3321                        |          | Las Condes | 300     | 912345      | 8333-5555     |
| 621        | Vivaceta 321                          |          | Recoleta   | 576     | 812345      | 9444-2222     |
| 620        | Recoleta 123                          |          | Santiago   | 300     | 912345      | 9555-5555     |
| 619        | 068 Samuel Islands Apt. 711, IN 45377 | [null]   | Joneston   | 1       | 17589       | (905)654-0273 |

Total rows: 10 of 10    Query complete 00:00:00.080    Ln 5, Col 34

```

1 SELECT * FROM public.staff
2 ORDER BY staff_id DESC LIMIT 5
3

```

| staff_id | first_name | last_name | address_id | email                         | store_id | active | username |
|----------|------------|-----------|------------|-------------------------------|----------|--------|----------|
| 4        | Juanito    | Staff2    | 621        | jstaff2@eval5.com             | 1        | true   | Juanito2 |
| 3        | Pedrito    | Staff1    | 620        | pstaff1@eval5.com             | 2        | true   | Pedrito1 |
| 2        | Jon        | Stephens  | 4          | Jon.Stephens@sakilastaff.c... | 2        | true   | Jon      |
| 1        | Mike       | Hillyer   | 3          | Mike.Hillyer@sakilastaff.com  | 1        | true   | Mike     |

### 3. OPERACIONES DE ELIMINACIÓN PARA LA ENTIDAD ACTOR

- **Eliminar un actor ("DELETE" ACTOR)**

Forma simple, eliminamos el actor haciendo una consulta a la tabla "actor".

La tabla no tiene restricciones ni FK de las que dependa.

Para este ejemplo, de los actores que creamos antes, usaremos el actor #203.

```

```sql
-- Eliminamos el registro del actor usando su actor_id

DELETE FROM actor
WHERE actor_id = 203;

-- Se verifica con la siguiente consulta

SELECT * FROM actor
ORDER BY actor_id DESC LIMIT 10
```

```

```

39
40
41   SELECT f_eliminar_empleado(5);
42
43   -- Eliminamos el registro del actor usando su actor_id
44
45   DELETE FROM actor
46   WHERE actor_id = 203; ← Red arrow here
47
48   -- Se verifica con la siguiente consulta
49
50   SELECT * FROM actor
51   ORDER BY actor_id DESC LIMIT 10

```

Data Output Messages Notifications

|   | actor_id | first_name | last_name | last_update                |
|---|----------|------------|-----------|----------------------------|
| 1 | 202      | Pedrito    | Actor2    | 2024-11-11 17:05:56.267736 |
| 2 | 201      | Juanito    | Actor1    | 2024-11-11 17:05:56.267736 |
| 3 | 200      | Thora      | Temple    | 2013-05-26 14:47:57.62     |
| 4 | 199      | Julia      | Fawcett   | 2013-05-26 14:47:57.62     |
| 5 | 198      | Mary       | Keitel    | 2013-05-26 14:47:57.62     |
| 6 | 197      | Reese      | West      | 2013-05-26 14:47:57.62     |
| 7 | 196      | Bela       | Walken    | 2013-05-26 14:47:57.62     |

Total rows: 10 of 10    Query complete 00:00:00.079    Ln 51, Col 32

138 AM

Para el siguiente caso, usando una función, eliminaremos un actor pasando como parámetro el número de su id (actor\_id). No tiene dependencias.

Eliminaremos el registro #202 de la tabla actor.

- **Eliminar la función (DROP)**

```

```sql
DROP FUNCTION IF EXISTS f_eliminar_actor(integer);
```

```

- **FUNCTION: f\_eliminar\_actor()**

```

```sql
CREATE OR REPLACE FUNCTION f_eliminar_actor(_actor_id integer)
RETURNS void AS $$
BEGIN
DELETE FROM actor
WHERE actor_id = _actor_id;
END;
$$
LANGUAGE plpgsql;
```

```

- **EJECUCIÓN**

The screenshot shows a PostgreSQL query editor with the following code:

```

34   WHERE address_id = _address_id;
35
36   END IF;
37
38   $$ LANGUAGE plpgsql;
39
40   SELECT f_eliminar_actor(202);
41
42   -- Eliminamos el registro del actor usando su actor_id
43
44   DELETE FROM actor
45   WHERE actor_id = 203;
46
47

```

A red arrow points from the left margin to the line `SELECT f\_eliminar\_actor(202);` which is circled in red.

Below the code, the Data Output tab shows the message: "Query returned successfully in 76 msec."

```

```sql
SELECT f_eliminar_actor(202);

-- Se verifica con la siguiente consulta

SELECT * FROM actor
ORDER BY actor_id DESC LIMIT 10
```

```

The screenshot shows a PostgreSQL data grid with the following table structure:

|   | actor_id     | first_name             | last_name              | last_update                 |
|---|--------------|------------------------|------------------------|-----------------------------|
|   | [PK] integer | character varying (45) | character varying (45) | timestamp without time zone |
| 1 | 204          | Dieguito               | Actor3                 | 2024-11-12 11:35:13.639429  |
| 2 | 201          | Juanito                | Actor1                 | 2024-11-11 17:05:56.267736  |
| 3 | 200          | Thora                  | Temple                 | 2013-05-26 14:47:57.62      |
| 4 | 199          | Julia                  | Fawcett                | 2013-05-26 14:47:57.62      |
| 5 | 198          | Mary                   | Keitel                 | 2013-05-26 14:47:57.62      |
| 6 | 197          | Reese                  | West                   | 2013-05-26 14:47:57.62      |
| 7 | 196          | Bela                   | Walken                 | 2013-05-26 14:47:57.62      |

## PROCEDIMIENTOS GENERALES DE ACTUALIZACIÓN (UPDATE)

### 1. OPERACIONES DE ACTUALIZACIÓN PARA LA ENTIDAD CUSTOMER

- **Actualizar un cliente ("UPDATE" CUSTOMER)**

Se proponen 3 métodos para actualizar los registros

- Forma simple.
- Función con un parámetro
- Función 2 con múltiples parámetros

**Forma simple**, usando un select con update y set, podemos actualizar un registro en la tabla customer, pero requerirá una acción por cada modificación.

## Sintáxis:

```
```sql
UPDATE <nombre_tabla>
SET <nombre_campo>= <valor_campo>
WHERE <condicion>;
```

```

Campos actualizables o modificables:

- store\_id
  - first\_name
  - last\_name
  - email
  - address\_id
  - activebool
  - active

Primero haremos una actualización simple por el ID de customer. Se modificará el correo del "customer\_id" = 602, de "mymail" a "testupdate".

```
```sql
UPDATE customer
SET email = 'jcordova@testupdate.com'
WHERE customer_id = 602;
```

```
219 -- Haremos un update manual del registro customer_id = 602
220 -- Cambiaremos el correo "jorge.cordova@mymail.com", por el correo "jcordova@testupdate.com"
221 UPDATE customer
222     SET
223         email = 'jcordova@testupdate.com'
224     WHERE customer_id = 602;
```

UPDATE #602

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp with time zone
1	607	1	Uso de Procedure UPDATE	Test Brady	TereGarcia@example.net	619	true	2024-09-27	2024-09-30 10:00:00
2	606	1	Test#2 uso de fx update	Gonzalez Test	cindystest2_updated@example.com	618	true	2024-09-27	2024-09-30 10:00:00
3	602	1	Jorge	Cordova	jorge.cordova@mymail.com	614	true	2024-09-25	2024-09-25 10:00:00
4	599	2	Austin	Cintron	austin.cintron@sakilacustomer.org	605	true	2006-02-14	2013-05-26 10:00:00
5	598	1	Wade	Delvalle	wade.delvalle@sakilacustomer.org	604	true	2006-02-14	2013-05-26 10:00:00

Total rows: 5 of 5    Query complete 00:00:00.168    Ln 224, Col 28

Ejecutando una sentencia "UPDATE", modificamos el correo del cliente.

```

91
92   SELECT * FROM customer
93   ORDER BY customer_id DESC LIMIT
94
95   -- La comprobación no es completa sin listar los cambios de la tabla "address".
96   -- Listamos las address de los últimos 5 registros agregados
97
98   SELECT * FROM address
99   ORDER BY address_id DESC LIMIT 5
100
101  -- Para indagar sobre las restricciones utilice la siguiente consulta

```

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp
1	607	1	Uso de Procedure UPDATE	Test Brady	TereGarcia@example.net	619	true	2024-09-27	2024-09-30
2	606	1	Test2 uso de fx update	Gonzalez Test	cindytest2_updated@example.com	618	true	2024-09-27	2024-09-30
3	602	1	Jorge	Cordova	jcordova@testupdate.com	614	true	2024-09-25	2024-09-30
4	599	2	Austin	Cintron	austin.cintron@sakilacustomer.org	605	true	2006-02-14	2013-05-26
5	598	1	Wade	Delvalle	wade.delvalle@sakilacustomer.org	604	true	2006-02-14	2013-05-26

Total rows: 5 of 5    Query complete 00:00:00.085    Ln 93, Col 34

- **FUNCIÓN 1: Actualiza cliente pasando un parámetro de tipo texto**

Párametros:

- "\_customer\_id" : El id del customer, de tipo entero.
- "\_columna" : El nombre de una columna cualquiera, tipo texto.
- "\_valor" : Una cadena de texto.

Retorna: Un mensaje de confirmación por consola.

Excepción: Esta versión sólo es para campos de tipo texto, se puede hacer una variante, modificando el tipo de dato por boolean, numeric, integer

Para evitar esto, que no es tan eficiente, en la siguiente versión, se agregaría la lógica usando JSONB, para que recibe argumentos múltiples.

```

```sql
-- Versión una columna dinámica
CREATE OR REPLACE FUNCTION f_actualiza_cliente(
    _customer_id INT,
    _columna TEXT,
    _valor TEXT
)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    -- Arma la consulta de actualización en forma dinámica
    EXECUTE format("UPDATE customer SET %I = %L WHERE customer_id = %L", _columna, _valor, _customer_id);

    -- Confirma que la actualización se realizó

```

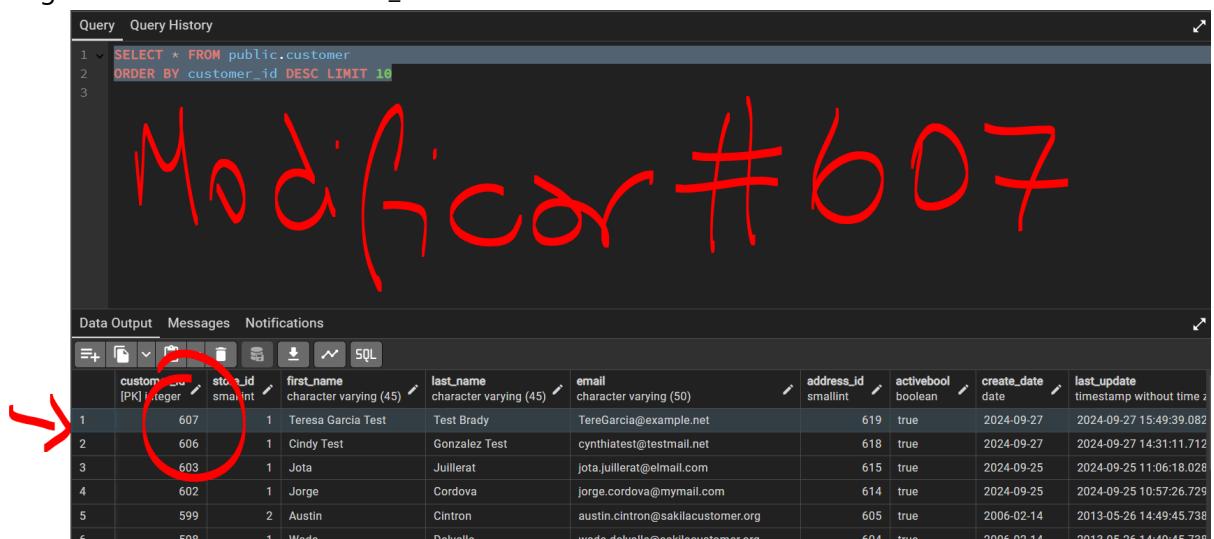
```

        RAISE NOTICE "Customer ID: %, columna % actualizada con valor %",
        _customer_id, _columna, _valor;
      END;
    $$;
  ```

``sql
-- EJECUCIÓN
SELECT f_actualiza_cliente(607, 'first_name', 'Uso de Función
UPDATE');
```

```

Registro a modificar en el "first\_name"



Query    Query History

```

1. SELECT * FROM public.customer
2. ORDER BY customer_id DESC LIMIT 10
3.

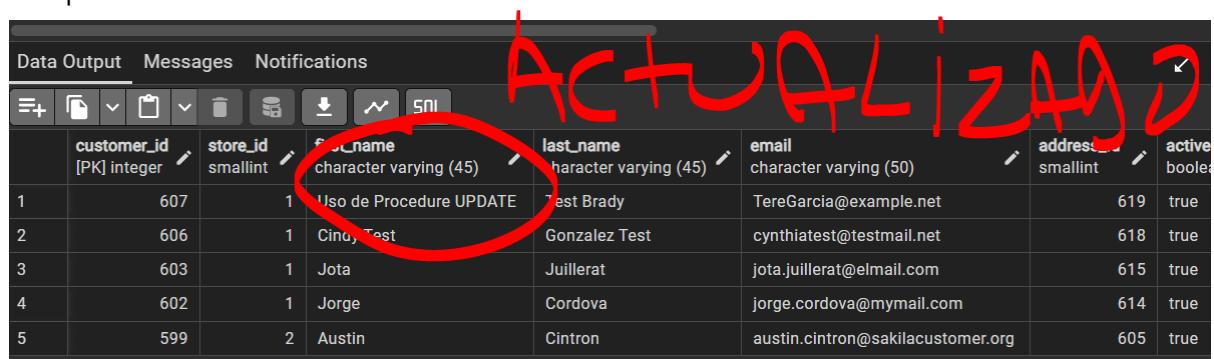
```

Modificar #607

Data Output    Messages    Notifications

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	active bool boolean	create_date date	last_update timestamp without time zone
1	607	1	Teresa Garcia Test	Test Brady	TereGarcia@example.net	619	true	2024-09-27	2024-09-27 15:49:39.082
2	606	1	Cindy Test	Gonzalez Test	cynthiatest@testmail.net	618	true	2024-09-27	2024-09-27 14:31:11.712
3	603	1	Jota	Juillerat	jota.juillerat@elmail.com	615	true	2024-09-25	2024-09-25 11:06:18.028
4	602	1	Jorge	Cordova	jorge.cordova@mymail.com	614	true	2024-09-25	2024-09-25 10:57:26.729
5	599	2	Austin	Cintron	austin.cintron@sakilacustomer.org	605	true	2006-02-14	2013-05-26 14:49:45.738
6	598	1	Wardle	Dalvalla	ward.dalvalla@sakilacustomer.org	604	true	2006-02-14	2013-05-26 14:49:45.738

Campo modificado con la cadena "Uso de Función UPDATE"



Data Output    Messages    Notifications

Actualizar

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	active bool boolean
1	607	1	Uso de Procedure UPDATE	Test Brady	TereGarcia@example.net	619	true
2	606	1	Cindy Test	Gonzalez Test	cynthiatest@testmail.net	618	true
3	603	1	Jota	Juillerat	jota.juillerat@elmail.com	615	true
4	602	1	Jorge	Cordova	jorge.cordova@mymail.com	614	true
5	599	2	Austin	Cintron	austin.cintron@sakilacustomer.org	605	true

- **FUNCIÓN 2: Actualiza cliente pasando parámetros múltiples**

Párametros:

- `_customer_id` : El id del customer, de tipo entero.
- `_updates JSONB` : Un conjunto de pares clave:valor

Retorna: Un mensaje de confirmación por consola.

```

``sql
-- Versión múltiples campos usando JSONB

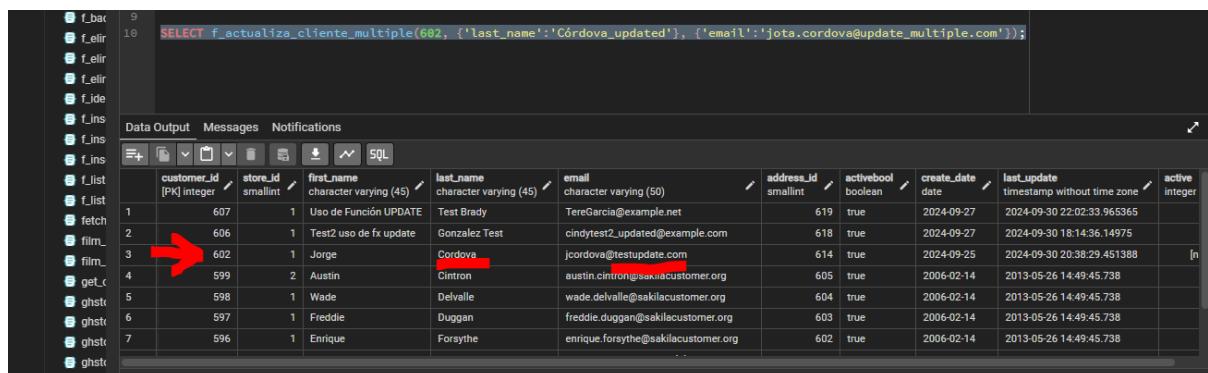
```

```

CREATE OR REPLACE FUNCTION f_actualiza_cliente_multiple(
    _customer_id INT,
    _updates JSONB
)
RETURNS TEXT
LANGUAGE plpgsql
AS
$$
DECLARE
    _columna TEXT;
    _valor TEXT;
    _set_clauses TEXT := '';
BEGIN
    -- Invento para armar el SET de una consulta con múltiples
    -- columnas dinámicamente
    FOR _columna, _valor IN SELECT * FROM jsonb_each_text(_updates)
    LOOP
        _set_clauses := _set_clauses || format('%I = %L, ', _columna,
        _valor);
    END LOOP;
    -- Se eliminan caracteres como la última coma y espacio extra al
    -- formar la consulta
    _set_clauses := rtrim(_set_clauses, ', ');
    -- Ejecutamos la actualización para más de 1 columna
    EXECUTE format('UPDATE customer SET %s WHERE customer_id = %L',
    _set_clauses, _customer_id);
    -- Mensaje con confirmación
    RETURN format('Customer ID %s: columnas actualizadas con los
    valores %s', _customer_id, _set_clauses);
END;
$$;
```
```
-- EJECUCIÓN
SELECT f_actualiza_cliente_multiple(602, '{"last_name": "Cordova_updated", "email": "jota.cordova@update_multiple.com"}');
```

```

### Vista antes de usar la función



|                                                            | customer_id | store_id | first_name            | last_name     | email                               | address_id | activebool | create_date | last_update                | active |
|------------------------------------------------------------|-------------|----------|-----------------------|---------------|-------------------------------------|------------|------------|-------------|----------------------------|--------|
| 1                                                          | 607         | 1        | Uso de Función UPDATE | Test Brady    | TereGarcia@example.net              | 619        | true       | 2024-09-27  | 2024-09-30 22:02:33.965365 |        |
| 2                                                          | 606         | 1        | Test uso de fx update | Gonzalez Test | cindytst2_updated@example.com       | 618        | true       | 2024-09-27  | 2024-09-30 18:14:36.14975  |        |
| 3                                                          | 602         | 1        | Jorge                 | Cordova       | jcordova@testupdate.com             | 614        | true       | 2024-09-25  | 2024-09-30 20:38:29.451388 | [n]    |
| 4                                                          | 599         | 2        | Austin                | Ciltron       | austin.cintron@saclacustomer.org    | 605        | true       | 2006-02-14  | 2013-05-26 14:49:45.738    |        |
| 5                                                          | 598         | 1        | Wade                  | Delvalle      | wade.delvalle@sakilacustomer.org    | 604        | true       | 2006-02-14  | 2013-05-26 14:49:45.738    |        |
| 6                                                          | 597         | 1        | Freddie               | Duggan        | freddie.duggan@sakilacustomer.org   | 603        | true       | 2006-02-14  | 2013-05-26 14:49:45.738    |        |
| 7                                                          | 596         | 1        | Enrique               | Forsythe      | enrique.forsythe@sakilacustomer.org | 602        | true       | 2006-02-14  | 2013-05-26 14:49:45.738    |        |
| Total rows: 10 of 10 Query complete 00:00:00.104 Ls10 Col1 |             |          |                       |               |                                     |            |            |             |                            |        |

Ejecutamos la función

```
10 |
11 SELECT f_actualiza_cliente_multiple(602, '{"last_name": "Cordova_updated", "email": "jota.cordova@update_multiple.com"}')
```

Data Output Messages Notifications

f\_actualiza\_cliente\_multiple  
text

| 1 | Customer ID 602: columnas actualizadas con los valores email = 'jota.cordova@update_multiple.com', last_name = 'Cordova_upd... |
|---|--------------------------------------------------------------------------------------------------------------------------------|

## Vista después de usar la función

```
1 ORDER BY customer_id DESC LIMIT 20
2
3
4 SELECT * FROM customer
5 ORDER BY customer_id DESC LIMIT 10
6
7
8 SELECT * FROM address
9 ORDER BY address_id DESC LIMIT 5
10
11 SELECT f_actualiza_cliente_multiple(602, '{"last_name": "Cordova_updated", "email": "jota.cordova@update_multiple.com"}')
```

Data Output Messages Notifications

SQL

|   | customer_id [PK] integer | store_id smallint | firstName character varying(45) | lastName character varying(45) | email character varying(50)         | address_id smallint | activebool boolean | create_date date | last_update timestamp without time zone | active integer |
|---|--------------------------|-------------------|---------------------------------|--------------------------------|-------------------------------------|---------------------|--------------------|------------------|-----------------------------------------|----------------|
| 1 | 607                      | 1                 | Uso de Función UPDATE           | Test Brady                     | TereGarcia@example.net              | 619                 | true               | 2024-09-27       | 2024-09-30 22:02:33.965365              | n              |
| 2 | 606                      | 1                 | Test uso de fx update           | Gonzalez Test                  | cindytest2_updated@example.com      | 618                 | true               | 2024-09-27       | 2024-09-30 18:14:36.14975               | n              |
| 3 | 602                      | 1                 | Jorge                           | Cordova_updated                | jota.cordova@update_multiple.com    | 614                 | true               | 2024-09-25       | 2024-11-12 19:30:24.978603              | n              |
| 4 | 599                      | 2                 | Austin                          | Cimtron                        | austin.cimtron@sakilacustomer.org   | 605                 | true               | 2006-02-14       | 2013-05-26 14:49:45.738                 | n              |
| 5 | 598                      | 1                 | Wade                            | Delvalle                       | wade.delvalle@sakilacustomer.org    | 604                 | true               | 2006-02-14       | 2013-05-26 14:49:45.738                 | n              |
| 6 | 597                      | 1                 | Freddie                         | Duggen                         | freddie.duggan@sakilacustomer.org   | 603                 | true               | 2006-02-14       | 2013-05-26 14:49:45.738                 | n              |
| 7 | 596                      | 1                 | Enrique                         | Forsythe                       | enrique.forsythe@sakilacustomer.org | 602                 | true               | 2006-02-14       | 2013-05-26 14:49:45.738                 | n              |

## 2. OPERACIONES DE ACTUALIZACIÓN PARA LA ENTIDAD STAFF

#### ○ Actualizar un empleado ("UPDATE" STAFF)

Al igual que con "customer", se puede utilizar cualquiera de los 3 métodos ya descritos para actualizar los registros de esta tabla

- Forma simple.
  - Función con un parámetro
  - Función 2 con múltiples parámetros

**Forma simple**, usando un select con update y set, podemos actualizar un registro en la tabla customer, pero requerirá una acción por cada modificación.

## Sintáxis:

```
```sql
UPDATE <nombre_tabla>
SET <nombre_campo>= <valor_campo>
WHERE <condicion>;
```

Campos actualizables o modificables:

- first\_name
  - last\_name
  - address\_id

- email
- store\_id
- active
- username
- password
- picture

Primero haremos una actualización simple por el ID del empleado. Se modificará el correo del "staff\_id" = 4, de "jstaff2" a "juanito\_staff2".

```
```sql
UPDATE customer
SET email = 'juanito_staff2@eval5.com'
WHERE staff_id = 4;
```
```

#### Vista antes de actualizar

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50) | store_id<br>smallint | active<br>boolean | username<br>character varying (16) | password<br>character varying (40) |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|---------------------------------|----------------------|-------------------|------------------------------------|------------------------------------|
| 1 | 4                        | Juanito                              | Staff2                              | 621                    | jstaff2@eval5.com               | 1                    | true              | Juanito2                           | pass123                            |
| 2 | 3                        | Pedrito                              | Staff1                              | 620                    | pstaff1e@eval5.com              | 2                    | true              | Pedrito1                           | pass123                            |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.c...   | 2                    | true              | Jon                                | 8cb2237d0679ca88db6464eac          |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com    | 1                    | true              | Mike                               | 8cb2237d0679ca88db6464eac          |

#### Vista después de actualizar

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50) | store_id<br>smallint | active<br>boolean | username<br>character varying (16) | password<br>character varying (40) |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|---------------------------------|----------------------|-------------------|------------------------------------|------------------------------------|
| 1 | 4                        | Juanito                              | Staff2                              | 621                    | juanito_staff2@eval5.com        | 1                    | true              | Juanito2                           | pass123                            |
| 2 | 3                        | Pedrito                              | Staff1                              | 620                    | pstaff1e@eval5.com              | 2                    | true              | Pedrito1                           | pass123                            |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.c...   | 2                    | true              | Jon                                | 8cb2237d0679ca88db6464eac          |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com    | 1                    | true              | Mike                               | 8cb2237d0679ca88db6464eac          |

#### ○ FUNCIÓN 1: Actualiza empleado pasando un parámetro de tipo texto

Párametros:

- "\_staff\_id" : El id del customer, de tipo entero.
- "\_columna" : El nombre de una columna cualquiera, que sea de tipo texto.
- "\_valor" : Una cadena de texto.

Retorna: Un mensaje de confirmación por consola.

Excepción: Esta versión sólo es para campos de tipo texto, se puede hacer una variante, modificando el tipo de dato por boolean, numeric, integer

Para evitar esto, que no es tan eficiente, en la siguiente versión, se agregará la lógica usando JSONB, para que recibe argumentos múltiples.

```
```sql
-- Versión una columna dinámica
CREATE OR REPLACE FUNCTION f_actualiza_empleado(
    _staff_id INT,
    _columna TEXT,
    _valor TEXT
)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    -- Arma la consulta de actualización en forma dinámica
    EXECUTE format('UPDATE staff SET %I = %L WHERE staff_id = %L',
    _columna, _valor, _staff_id);
    -- Confirma que la actualización se realizó
    RAISE NOTICE '%', format('Staff ID: %s, columna %s actualizada con
    valor %s', _staff_id, _columna, _valor);
END;
$$;
```

```

## Eliminar función

```
```sql
DROP FUNCTION IF EXISTS f_actualiza_empleado(integer, text, text);
```

```

## Ejecución

```
```sql
SELECT f_actualiza_empleado(3, 'first_name', 'Pedro Test Función
UPDATE');
```

```

Registro a modificar en el "first\_name"

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50) | store_id<br>smallint | active<br>boolean | username<br>character varying (16) |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|---------------------------------|----------------------|-------------------|------------------------------------|
| 1 | 4                        | Juanito                              | Staff2                              | 621                    | juanito_staff2@eval5.com        | 1                    | true              | Juanito2                           |
| 2 | 3                        | Pedrito                              | Staff1                              | 620                    | pstaff1e@eval5.com              | 2                    | true              | Pedrito1                           |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.c...   | 2                    | true              | Jon                                |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com    | 1                    | true              | Mike                               |

### Campo modificado con la cadena "Uso de Función UPDATE"

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50) | store_id<br>smallint | active<br>boolean | username<br>character varying (16) |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|---------------------------------|----------------------|-------------------|------------------------------------|
| 1 | 4                        | Juanito                              | Staff2                              | 621                    | juanito_staff2@eval5.com        | 1                    | true              | Juanito2                           |
| 2 | 3                        | Pedro Test Función UPDATE            | Staff1                              | 620                    | pstaff1e@eval5.com              | 2                    | true              | Pedrito1                           |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.c...   | 2                    | true              | Jon                                |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com    | 1                    | true              | Mike                               |

- **FUNCIÓN 2: Actualiza empleados (staff) pasando parámetros múltiples**

Párametros:

- `_staff_id` : El id del customer, de tipo entero.
- `_updates JSONB` : Un conjunto de pares clave:valor

Retorna: Un mensaje de confirmación por consola.

```
```sql
-- Versión múltiples campos usando JSONB
CREATE OR REPLACE FUNCTION f_actualiza_empleado_multiple(
    _staff_id INT,
    _updates JSONB
)
RETURNS TEXT
LANGUAGE plpgsql
AS
$$
DECLARE
    _columna TEXT;
    _valor TEXT;
    _set_clauses TEXT := '';
BEGIN
    -- Invento para armar el SET de una consulta con múltiples
    -- columnas dinámicamente
    FOR _columna, _valor IN SELECT * FROM jsonb_each_text(_updates)
    LOOP
        _set_clauses := _set_clauses || format('%I = %L, ', _columna,
    _valor);
    END LOOP;
    -- Se eliminan caracteres como la última coma y espacio extra al
    -- formar la consulta
    _set_clauses := rtrim(_set_clauses, ', ');
    -- Ejecutamos la actualización para más de 1 columna
    EXECUTE format('UPDATE staff SET %s WHERE staff_id = %L',
    _set_clauses, _staff_id);
    -- Mensaje con confirmación
    RETURN format('Staff ID %s: columnas actualizadas con los valores
    %s', _staff_id, _set_clauses);
END;
$$;
```

```

```
```sql
-- EJECUCIÓN
SELECT f_actualiza_empleado_multiple(4, '{"last_name": "Staff2_Múltiple", "email": "juanito_staff2_test_multiple@eval5.com"}');
```

```

### Vista antes de usar la función

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50) | store_id<br>smallint | active<br>boolean | username<br>character varying (16) |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|---------------------------------|----------------------|-------------------|------------------------------------|
| 1 | 4                        | Juanito                              | Staff2                              | 621                    | juanito_staff2@eval5.com        | 1                    | true              | Juanito2                           |
| 2 | 3                        | Pedro Test Función UPDATE            | Staff1                              | 620                    | pstaff1e@eval5.com              | 2                    | true              | Pedrito1                           |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.c...   | 2                    | true              | Jon                                |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com    | 1                    | true              | Mike                               |

### Ejecutamos la función

| Data Output                   | Messages   | Notifications |
|-------------------------------|--|---------------|
|                               |  |               |
| f_actualiza_empleado_multiple | text   |               |
| 1                             | Staff ID 4: columnas actualizadas con los valores email = 'juanito_staff2_test_multiple@eval5.com', last_name = 'Staff2_Múlt...' |               |

### Vista después de usar la función

|   | staff_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | address_id<br>smallint | email<br>character varying (50)        | store_id<br>smallint | active<br>boolean | username<br>character |
|---|--------------------------|--------------------------------------|-------------------------------------|------------------------|--|----------------------|-------------------|-----------------------|
| 1 | 4                        | Juanito                              | Staff2_Múltiple                     | 621                    | juanito_staff2_test_multiple@eval5.com | 1                    | true              | Juanito2              |
| 2 | 3                        | Pedro Test Función UPDATE            | Staff1                              | 620                    | pstaff1e@eval5.com                     | 2                    | true              | Pedrito1              |
| 3 | 2                        | Jon                                  | Stephens                            | 4                      | Jon.Stephens@sakilastaff.com           | 2                    | true              | Jon                   |
| 4 | 1                        | Mike                                 | Hillyer                             | 3                      | Mike.Hillyer@sakilastaff.com           | 1                    | true              | Mike                  |

## 3. OPERACIONES DE ACTUALIZACIÓN PARA LA ENTIDAD ACTOR

### ○ Actualizar un actor ("UPDATE" ACTOR)

Al igual que con los anteriores, mostraré 3 formas de actualizar los registros de la tabla "ACTOR".

- Forma simple.
- Función con un parámetro
- Función 2 con múltiples parámetros

Sintáxis:

```

```sql
UPDATE <nombre_tabla>

```

```

SET <nombre_campo>= <valor_campo>
WHERE <condicion>;
```

```

Campos actualizables o modificables:

- first\_name
- last\_name

Primero haremos una actualización simple por el ID del actor. Se modificará el nombre del "actor\_id" = 204, de "Dieguito" a "Diego Test Simple".

```

```sql
UPDATE actor
SET first_name = 'Diego Test Simple'
WHERE actor_id = 204;
```

```

Vista antes de usar la función

|   | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
|---|--------------------------|--------------------------------------|-------------------------------------|--|
| 1 | 204                      | Dieguito                             | Actor3                              | 2024-11-12 11:35:13.639429                 |
| 2 | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 3 | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |
| 4 | 199                      | Julia                                | Fawcett                             | 2013-05-26 14:47:57.62                     |
| 5 | 198                      | Mary                                 | Keitel                              | 2013-05-26 14:47:57.62                     |
| 6 | 197                      | Reese                                | West                                | 2013-05-26 14:47:57.62                     |
| 7 | 196                      | Bela                                 | Walken                              | 2013-05-26 14:47:57.62                     |

Ejecutamos la función

```

66
67 UPDATE actor
68     SET first_name = 'Diego Test Simple'
69     WHERE actor_id = 204;

```

Data Output    Messages    Notifications

UPDATE 1

Query returned successfully in 73 msec.

Vista después de usar la función

| Data Output Messages Notifications |                          |                                      |                                     |  |
|------------------------------------|--------------------------|--------------------------------------|-------------------------------------|--|
|                                    | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
| 1                                  | 204                      | Diego Test Simple                    | Actor3                              | 2024-11-13 00:41:50.083819                 |
| 2                                  | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 3                                  | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |
| 4                                  | 199                      | Julia                                | Fawcett                             | 2013-05-26 14:47:57.62                     |
| 5                                  | 198                      | Mary                                 | Keitel                              | 2013-05-26 14:47:57.62                     |
| 6                                  | 197                      | Reese                                | West                                | 2013-05-26 14:47:57.62                     |
| 7                                  | 196                      | Bela                                 | Walken                              | 2013-05-26 14:47:57.62                     |

- **FUNCIÓN 1: Actualiza un actor pasando un parámetro de tipo texto**

Párametros:

- "\_actor\_id" : El id del actor, de tipo entero.
- "\_columna" : El nombre de una columna cualquiera, que sea de tipo texto.
- "\_valor" : Una cadena de texto.

Retorna: Un mensaje de confirmación por consola.

Excepción: Esta versión sólo es para campos de tipo texto, se puede hacer una variante, modificando el tipo de dato por boolean, numeric, integer

Para evitar esto, que no es tan eficiente, en la siguiente versión, se agregará la lógica usando JSONB, para que reciba argumentos múltiples.

```
```sql
-- Versión una columna dinámica
CREATE OR REPLACE FUNCTION f_actualiza_actor(
    _actor_id INT,
    _columna TEXT,
    _valor TEXT
)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    -- Arma la consulta de actualización en forma dinámica
    EXECUTE format('UPDATE actor SET %I = %L WHERE actor_id = %L',
    _columna, _valor, _actor_id);

    -- Confirma que la actualización se realizó
    RAISE NOTICE '%', format('Actor ID: %s, columna %s actualizada con
    valor %s', _actor_id, _columna, _valor);
END;
```

```

\$\$;

三

## **Eliminar función**

```sql

```
DROP FUNCTION IF EXISTS f_actualiza_actor(integer, text, text);
```

—

## Ejecución

```sql

```
SELECT f_actualiza_actor(205, 'first_name', 'Jorge Test Función 1  
COL');
```

—

## Vista antes de usar la función

Data Output   Messages   Notifications

|   | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
|---|--------------------------|--------------------------------------|-------------------------------------|--|
| 1 | 206                      | Cenicienta                           | Actriz5                             | 2024-11-13 10:36:49.734066                 |
| 2 | 205                      | Jota                                 | Actor4                              | 2024-11-13 10:36:43.62373                  |
| 3 | 204                      | Diego Test Simple                    | Actor3                              | 2024-11-13 00:41:50.083819                 |
| 4 | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 5 | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |

Ejecutamos la función

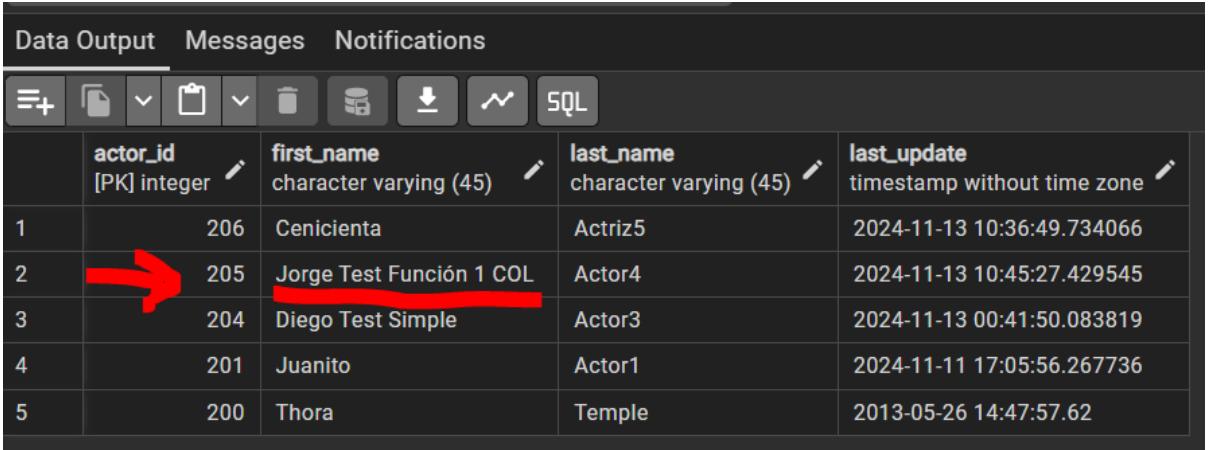
```
133
134 SELECT f_actualiza_actor(205, 'first_name', 'Jorge Test Función 1 COL');
```

Data Output Messages Notifications

f\_actualiza\_actor  
void

|   | f_actualiza_actor | lock |
|---|-------------------|------|
| 1 |                   |      |

Vista después de usar la función



|   | <b>actor_id</b><br>[PK] integer | <b>first_name</b><br>character varying (45) | <b>last_name</b><br>character varying (45) | <b>last_update</b><br>timestamp without time zone |
|---|---------------------------------|---|--|---|
| 1 | 206                             | Cenicienta                                  | Actriz5                                    | 2024-11-13 10:36:49.734066                        |
| 2 | 205                             | Jorge Test Función 1 COL                    | Actor4                                     | 2024-11-13 10:45:27.429545                        |
| 3 | 204                             | Diego Test Simple                           | Actor3                                     | 2024-11-13 00:41:50.083819                        |
| 4 | 201                             | Juanito                                     | Actor1                                     | 2024-11-11 17:05:56.267736                        |
| 5 | 200                             | Thora                                       | Temple                                     | 2013-05-26 14:47:57.62                            |

- **FUNCIÓN 2: Actualiza empleados (staff) pasando parámetros múltiples**

Párametros:

- `_actor_id` : El id del actor, de tipo entero.
- `_updates JSONB` : Un conjunto de pares clave:valor

Retorna: Un mensaje de confirmación por consola.

```
```sql
-- Versión múltiples campos usando JSONB
CREATE OR REPLACE FUNCTION f_actualiza_actor_multiple(
    _actor_id INT,
    _updates JSONB
)
RETURNS TEXT
LANGUAGE plpgsql
AS
$$
DECLARE
    _columna TEXT;
    _valor TEXT;
    _set_clauses TEXT := '';
BEGIN
    -- Invento para armar el SET de una consulta con múltiples
    -- columnas dinámicamente
    FOR _columna, _valor IN SELECT * FROM jsonb_each_text(_updates)
    LOOP
        _set_clauses := _set_clauses || format('%I = %L, ', _columna,
        _valor);
    END LOOP;
    -- Se eliminan caracteres como la última coma y espacio extra al
    -- formar la consulta
    _set_clauses := rtrim(_set_clauses, ', ');
    -- Ejecutamos la actualización para más de 1 columna
    EXECUTE format('UPDATE actor SET %s WHERE actor_id = %L',
    _set_clauses, _actor_id);

```

```
-- Mensaje con confirmación
RETURN format('Actor ID %s: columnas actualizadas con los valores
%s', _actor_id, _set_clauses);
END;
$$;
```

```

## Eliminar función

```
```sql
DROP FUNCTION IF EXISTS f_actualiza_actor_multiple(integer, jsonb);
```

```

## Ejecutar función

```
```sql
SELECT f_actualiza_actor_multiple(206, '{"first_name": "Sra.
Cenicienta", "last_name": "Actriz5 Test Función Múltiple"}');
```

```

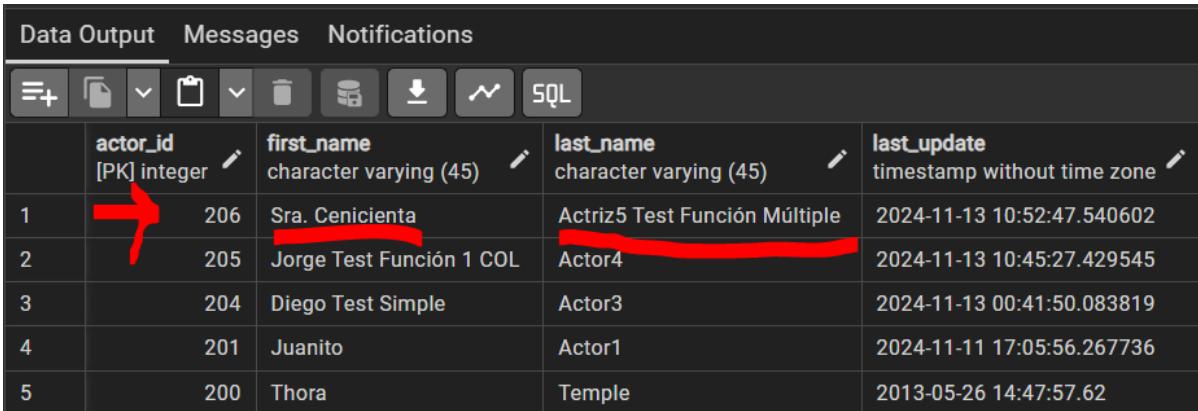
### Vista antes de usar la función

|   | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
|---|--------------------------|--------------------------------------|-------------------------------------|--|
| 1 | 206                      | Cenicienta                           | Actriz5                             | 2024-11-13 10:36:49.734066                 |
| 2 | 205                      | Jorge Test Función 1 COL             | Actor4                              | 2024-11-13 10:45:27.429545                 |
| 3 | 204                      | Diego Test Simple                    | Actor3                              | 2024-11-13 00:41:50.083819                 |
| 4 | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 5 | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |

### Ejecutamos la función

|   | f_actualiza_actor_multiple<br>text   |
|---|--|
| 1 | Actor ID 206: columnas actualizadas con los valores last_name = 'Actriz5 Test Función Múltiple', first_name = 'Sra. Cenici...' |

## Vista después de usar la función



|   | actor_id<br>[PK] integer | first_name<br>character varying (45) | last_name<br>character varying (45) | last_update<br>timestamp without time zone |
|---|--------------------------|--------------------------------------|-------------------------------------|--|
| 1 | 206                      | Sra. Cenicienta                      | Actriz5 Test Función Múltiple       | 2024-11-13 10:52:47.540602                 |
| 2 | 205                      | Jorge Test Función 1 COL             | Actor4                              | 2024-11-13 10:45:27.429545                 |
| 3 | 204                      | Diego Test Simple                    | Actor3                              | 2024-11-13 00:41:50.083819                 |
| 4 | 201                      | Juanito                              | Actor1                              | 2024-11-11 17:05:56.267736                 |
| 5 | 200                      | Thora                                | Temple                              | 2013-05-26 14:47:57.62                     |

## CONSULTAS ESPECÍFICAS (QUERIES)

## 1. CLIENTES QUE HAN RENTADO POR AÑO Y MES

## o Análisis

Esta consulta debe traer todos los clientes que han rentado películas para un determinado mes y año.

a) Para eficientar las consultas, primero que todo, podemos crear un índice en la tabla, sobre el 'campo rental\_date', esto mejorará el rendimiento de la consulta.

```
```sql
CREATE INDEX idx_rental_rental_date ON rental(rental_date); -- Crea
el índice si no existe, se ejecuta 1 vez.
```
```

b) Para no iterar entre pruebas y errores, buscando los años y meses que si tienen registros, primero, analizamos la cardinalidad de los arrendamientos por mes y año. Así podemos determinar qué períodos se pueden consultar y organizar mejor una consulta más eficiente.

```
```sql
SELECT
    EXTRACT(YEAR FROM rental_date) AS año,
    EXTRACT(MONTH FROM rental_date) AS mes,
    COUNT(*) AS cantidad_registros
FROM
    rental
GROUP BY
    año, mes
ORDER BY
    año, mes;
```
```

Data Output    Messages    Notifications

|   | año<br>numeric | mes<br>numeric | cantidad_registros<br>bigint |
|---|----------------|----------------|------------------------------|
| 1 | 2005           | 5              | 1156                         |
| 2 | 2005           | 6              | 2311                         |
| 3 | 2005           | 7              | 6709                         |
| 4 | 2005           | 8              | 5686                         |
| 5 | 2006           | 2              | 182                          |

c) Luego, podemos ir armando progresivamente nuestra consulta SQL simple. Para el ejemplo, la consulta trae todos los registros de arrendamientos, para un año y mes determinado, en este caso 2006 y mes 02, pero falta hacerlo para un "customer" específico.

```
```sql
SELECT rental.*, customer.*          -- Selecciona tablas rental y
customer
FROM rental                         -- Trae los datos de la tabla
rental
INNER JOIN customer ON rental.customer_id = customer.customer_id -- Une datos usando el id de cliente
WHERE DATE_PART('YEAR', rental_date) = 2006 AND      -- Filtra los
resultados por año
      DATE_PART('MONTH', rental_date) = 02           -- Filtra los
resultados por mes
ORDER BY rental_date;                  -- Ordena los
resultados por fecha
```

```

```

76 ✓ SELECT rental.*, customer.*      -- Selecciona tablas rental y customer
77   FROM rental                      -- Trae los datos de la tabla rental
78   INNER JOIN customer ON rental.customer_id = customer.customer_id -- Une datos usando el id de cliente
79   WHERE DATE_PART('YEAR', rental.rental_date) = 2006 AND      -- Filtra los resultados por año
80       DATE_PART('MONTH', rental.rental_date) = 02            -- Filtra los resultados por mes
81   ORDER BY rental.rental_date;

```

Data Output Messages Notifications

SQL

| rentalId | rental_date         | inventory_id | customer_id | return_date | staff_id | last_update         | customer_id | store_id | first_name |
|----------|---------------------|--------------|-------------|-------------|----------|---------------------|-------------|----------|------------|
| 1        | 2006-02-14 15:16:03 | 2047         | 155         | [null]      | 1        | 2006-02-16 02:30:53 | 155         | 1        | Gail       |
| 2        | 2006-02-14 15:16:03 | 2026         | 335         | [null]      | 1        | 2006-02-16 02:30:53 | 335         | 1        | Grego      |
| 3        | 2006-02-14 15:16:03 | 1556         | 479         | [null]      | 1        | 2006-02-16 02:30:53 | 479         | 1        | Zacha      |
| 4        | 2006-02-14 15:16:03 | 1545         | 83          | [null]      | 1        | 2006-02-16 02:30:53 | 83          | 1        | Louise     |
| 5        | 2006-02-14 15:16:03 | 4106         | 219         | [null]      | 2        | 2006-02-16 02:30:53 | 219         | 2        | Willie     |
| 6        | 2006-02-14 15:16:03 | 817          | 99          | [null]      | 1        | 2006-02-16 02:30:53 | 99          | 2        | Emily      |
| 7        | 2006-02-14 15:16:03 | 1857         | 192         | [null]      | 2        | 2006-02-16 02:30:53 | 192         | 1        | Laurie     |
| 8        | 2006-02-14 15:16:03 | 478          | 11          | [null]      | 2        | 2006-02-16 02:30:53 | 11          | 2        | Lisa       |
| 9        | 2006-02-14 15:16:03 | 1622         | 597         | [null]      | 2        | 2006-02-16 02:30:53 | 597         | 1        | Freddi     |
| 10       | 2006-02-14 15:16:03 | 3043         | 53          | [null]      | 2        | 2006-02-16 02:30:53 | 53          | 1        | Heath      |

Total rows: 182 of 182 Query complete 00:00:00.123 Ln 80, Col 44

## ○ Solución

La forma simple, es hacerlo a través de una consulta en la que indiquemos el "customer\_id", un año y mes específicos, de la siguiente manera:

```

```sql
SELECT
    rental.rental_id,
    rental.rental_date,
    rental.customer_id,
    customer.first_name,
    customer.last_name
FROM
    rental
INNER JOIN
    customer ON rental.customer_id = customer.customer_id
WHERE
    rental.customer_id = 300
    AND DATE_PART('YEAR', rental.rental_date) = 2005
    AND DATE_PART('MONTH', rental.rental_date) = 7
ORDER BY
    rental.rental_date ASC;
```

```

Lo cual retornará:

```

183 -- Consulta simple customer, año, mes
184 SELECT
185     rental.rental_id,
186     rental.rental_date,
187     rental.customer_id,
188     customer.first_name,
189     customer.last_name
190 FROM
191     rental
192 INNER JOIN
193     customer ON rental.customer_id = customer.customer_id
194 WHERE
195     rental.customer_id = 300
196     AND DATE_PART('YEAR', rental.rental_date) = 2005
197     AND DATE_PART('MONTH', rental.rental_date) = 7
198 ORDER BY
199     rental.rental_date ASC;

```

Data Output Messages Notifications

|   | rental_id | rental_date         | customer_id | first_name | last_name  |
|---|-----------|---------------------|-------------|------------|------------|
| 1 | 3775      | 2005-07-06 13:27:33 | 300         | John       | Farnsworth |
| 2 | 4030      | 2005-07-07 02:25:42 | 300         | John       | Farnsworth |
| 3 | 5562      | 2005-07-10 03:17:42 | 300         | John       | Farnsworth |
| 4 | 5705      | 2005-07-10 10:09:17 | 300         | John       | Farnsworth |
| 5 | 6111      | 2005-07-11 07:26:57 | 300         | John       | Farnsworth |

Total rows: 14 of 14    Query complete 00:00:00.077    Ln 199, Col 28

Una solución más elaborada, es usar una función que retorne la consulta. La siguiente función, "f\_listar\_por\_customer\_anno\_y\_mes()", permite pasar como parámetros, el valor del "customer id", un año y mes específicos.

**Recibe :** Valores enteros para el "customer\_id", año y mes.

Se recomienda hacer algunas consultas previas para conocer el rango de fechas y la cardinalidad de los meses para saber si lo que retorna la consulta es correcto Y COHERENTE.

**Retorna :** Hace una unión entre "rental" y la tabla "customer", a través del id del cliente y trae los registros de la tabla "rental" (arrendamiento de películas) para un año y mes específico, para un cliente particular.

```

```sql
CREATE OR REPLACE FUNCTION f_listar_por_customer_anno_y_mes(
    _customer_id INT,
    _year INT,
    _month INT
)
RETURNS TABLE (
    rental_id INT,
    rental_date TIMESTAMP WITHOUT TIME ZONE,
    customer_id INT,
    first_name character varying,
    last_name character varying
)
AS $$
BEGIN
    RETURN QUERY

```

```
SELECT
    r.rental_id,
    r.rental_date,
    c.customer_id,
    c.first_name,
    c.last_name
FROM rental AS r
INNER JOIN customer AS c ON r.customer_id = c.customer_id
WHERE r.customer_id = _customer_id
    AND DATE_PART('YEAR', r.rental_date) = _year
    AND DATE_PART('MONTH', r.rental_date) = _month
ORDER BY r.rental_date;
END;
$$
LANGUAGE plpgsql;
```
```

- **Eliminar la función**

```
```sql
DROP FUNCTION IF EXISTS f_listar_por_customer_anno_y_mes(integer,
integer, integer);
```
```

- **Ejecutar la función**

Como retorna un json, le anteponemos "\* from", para que desempaque el diccionario y lo muestre como una tabla separada por columnas.

```
```sql
SELECT * from f_listar_por_customer_anno_y_mes(300, 2005, 7);
```
```

- **Vista de la consulta usando la función**

|    | rental_id<br>integer | rental_date<br>timestamp without time zone | customer_id<br>integer | first_name<br>character varying | last_name<br>character varying |
|----|----------------------|--|------------------------|---------------------------------|--------------------------------|
| 1  | 3775                 | 2005-07-06 13:27:33                        | 300                    | John                            | Farnsworth                     |
| 2  | 4030                 | 2005-07-07 02:25:42                        | 300                    | John                            | Farnsworth                     |
| 3  | 5562                 | 2005-07-10 03:17:42                        | 300                    | John                            | Farnsworth                     |
| 4  | 5705                 | 2005-07-10 10:09:17                        | 300                    | John                            | Farnsworth                     |
| 5  | 6111                 | 2005-07-11 07:26:57                        | 300                    | John                            | Farnsworth                     |
| 6  | 6822                 | 2005-07-12 18:23:39                        | 300                    | John                            | Farnsworth                     |
| 7  | 6998                 | 2005-07-27 01:16:29                        | 300                    | John                            | Farnsworth                     |
| 8  | 7815                 | 2005-07-28 08:14:11                        | 300                    | John                            | Farnsworth                     |
| 9  | 8117                 | 2005-07-28 19:20:16                        | 300                    | John                            | Farnsworth                     |
| 10 | 8210                 | 2005-07-28 23:31:05                        | 300                    | John                            | Farnsworth                     |
| 11 | 8283                 | 2005-07-29 01:52:22                        | 300                    | John                            | Farnsworth                     |
| 12 | 9078                 | 2005-07-30 08:01:00                        | 300                    | John                            | Farnsworth                     |
| 13 | 9127                 | 2005-07-30 09:46:36                        | 300                    | John                            | Farnsworth                     |
| 14 | 9791                 | 2005-07-31 10:35:22                        | 300                    | John                            | Farnsworth                     |

## 2. LISTAR TODOS LOS PAGOS ACUMULADOS POR FECHA

- **Análisis**

Esta consulta debe traer la cantidad y el monto total de pagos por fecha. Por ejemplo, el día 1/1/2005, se hicieron 10 ventas, que totalizaron \$80,99.

- **Consulta simple**

La forma sencilla de resolverlo es a través de una consulta que agrupe por fecha, las ventas del día, esta es la consulta:

```
```sql
-- Listar acumulado de payments por fecha
SELECT
    CAST(payment_date AS DATE) AS "Fecha de Pago",
    COUNT(*) AS "Cantidad de Pagos",
    SUM(amount) AS "Acumulado Venta"
FROM payment
GROUP BY CAST(payment_date AS DATE)
ORDER BY CAST(payment_date AS DATE);
```
```

- **Vista de la consulta simple**

```

118  SELECT
119      CAST(payment_date AS DATE) AS "Fecha de Pago",
120      COUNT(*) AS "Cantidad de Pagos",
121      SUM(amount) AS "Acumulado Venta"
122  FROM payment
123  GROUP BY CAST(payment_date AS DATE)
124  ORDER BY CAST(payment_date AS DATE);

```

Data Output Messages Notifications

|    | Fecha de Pago<br>date | Cantidad de Pagos<br>bigint | Acumulado Venta<br>numeric |
|----|-----------------------|-----------------------------|----------------------------|
| 1  | 2007-02-14            | 27                          | 116.73                     |
| 2  | 2007-02-15            | 308                         | 1188.92                    |
| 3  | 2007-02-16            | 282                         | 1154.18                    |
| 4  | 2007-02-17            | 283                         | 1188.17                    |
| 5  | 2007-02-18            | 302                         | 1275.98                    |
| 6  | 2007-02-19            | 310                         | 1290.90                    |
| 7  | 2007-02-20            | 291                         | 1219.09                    |
| 8  | 2007-02-21            | 213                         | 917.87                     |
| 9  | 2007-03-01            | 676                         | 2808.24                    |
| 10 | 2007-03-02            | 595                         | 2550.05                    |
| 11 | 2007-03-16            | 72                          | 299.28                     |
| 12 | 2007-03-17            | 584                         | 2442.16                    |
| 13 | 2007-03-18            | 624                         | 2701.76                    |

Total rows: 32 of 32    Query complete 00:00:00.072    Ln 119, Col 12

- Function: **f\_listar\_acumulado\_ventas\_fecha()**

Trataremos de proponer una solución más elaborada, que permita recibir parámetros como el año o mes y, por defecto, traiga todas las ventas de la tabla "payment".

```

```sql
CREATE OR REPLACE FUNCTION f_listar_acumulado_ventas_fecha(
    p_year INT DEFAULT NULL
)
RETURNS TABLE (
    payment_date DATE,
    v_qty_pagos BIGINT,
    v_amount_pagos NUMERIC
)
AS $$
BEGIN
    RETURN QUERY
        SELECT CAST(p.payment_date AS DATE),
               COUNT(*) AS v_qty_pagos,
               SUM(p.amount) AS v_amount_pagos
        FROM payment AS p

```

```

    WHERE p_year IS NULL OR EXTRACT(YEAR FROM p.payment_date) = p_year
    GROUP BY CAST(p.payment_date AS DATE)
    ORDER BY CAST(p.payment_date AS DATE);
END;
$$
LANGUAGE plpgsql;
```

```

- **Eliminar la función**

```

```sql
DROP FUNCTION IF EXISTS f_listar_acumulado_ventas_fecha(integer);
```

```

- **Vista de la consulta usando la función, por defecto toma todo el período**

```

153  SELECT * FROM f_listar_acumulado_ventas_fecha();
154
155  -- Listar pr fecha qtty y monto ventas 2 (corregida)
156  CREATE OR REPLACE FUNCTION f_listar_acumulado_ventas_fecha(
157      p_year INT DEFAULT NULL
158  )
159  RETURNS TABLE (
160      payment_date DATE,
161      v_qty_pagos BIGINT,
162      v_amount_pagos NUMERIC
163  )
164  AS $$$
165  BEGIN
166      RETURN QUERY
167      SELECT CAST(p.payment_date AS DATE),
168              COUNT(*) AS v_qty_pagos,
169              SUM(p.amount) AS v_amount_pagos
170      FROM payment AS p
171      WHERE p.year IS NULL OR EXTRACT(YEAR FROM p.payment_date) = p.year

```

Data Output    Messages    Notifications

|   | payment_date<br>date | v_qty_pagos<br>bigint | v_amount_pagos<br>numeric |
|---|----------------------|-----------------------|---------------------------|
| 1 | 2007-02-14           | 27                    | 116.73                    |
| 2 | 2007-02-15           | 308                   | 1188.92                   |
| 3 | 2007-02-16           | 282                   | 1154.18                   |
| 4 | 2007-02-17           | 283                   | 1188.17                   |

Total rows: 32 of 32    Query complete 00:00:00.072    Ln 154, Col 1

Si probamos con otros años, por ejemplo año=2006, dentro del rango de fechas, se verifica que no hay datos, es decir, la consulta no retorna registros.

```

152
153     SELECT * FROM f_listar_acumulado_ventas_fecha();
154     SELECT * FROM f_listar_acumulado_ventas_fecha(2006);
155
156     -- Listar pr fecha qtty y monto ventas 2 (corregida)
157     CREATE OR REPLACE FUNCTION f_listar_acumulado_ventas_fecha()
  
```

Data Output Messages Notifications

|  | payment_date | v_qty_pagos | v_amount_pagos |
|--|--------------|-------------|----------------|
|  | date         | bigint      | numeric        |

Se puede verificar que sólo 2007 tiene datos mirando la tabla o, con una consulta como la siguiente, que muestre qué mes y año tiene ventas, para el caso, comprobamos que no hay más datos, lo que también se confirma al mirar la tabla "payment":

```

193 v  SELECT
194      EXTRACT(YEAR FROM payment_date) AS year,
195      EXTRACT(MONTH FROM payment_date) AS month,
196      COUNT(*) AS total_ventas
197  FROM
198      payment
199  GROUP BY
200      EXTRACT(YEAR FROM payment_date),
201      EXTRACT(MONTH FROM payment_date);
  
```

Data Output Messages Notifications

|   | year | month | total_ventas |
|---|------|-------|--------------|
| 1 | 2007 | 4     | 6754         |
| 2 | 2007 | 5     | 182          |
| 3 | 2007 | 2     | 2016         |
| 4 | 2007 | 3     | 5644         |

Total rows: 4 of 4    Query complete 00:00:00.063    Ln 197, Col 5

### 3. LISTAR TODOS LOS FILMS DEL AÑO 2006, QUE TENGAN UNA CALIFICACIÓN MAYOR A 4.0

- **Análisis**

Esta consulta SQL debe permitir obtener una lista de todas las películas del año 2006 que tengan una calificación superior a 4.

Como no hay una especificación, y para simplificar la vista, asumiremos que los campos requeridos son:

- film\_id: Identificador único de la película.
- title: Título de la película.
- release\_year: Año de lanzamiento de la película.
- rental\_rate: Calificación de la película.

- **Consulta simple**

En forma directa y sencilla, generamos un "select" a la tabla "film", indicando los campos que queremos ver, a los que les hemos dado un "nombre semántico" o más entendible.

```
```sql
SELECT film_id AS "id película", title AS "Título del film",
release_year AS "Año de estreno", rental_rate AS "Calificación"
FROM film
WHERE release_year = 2006
AND rental_rate > 4;
```
```

- **Vista de la consulta simple**

```
203 -- Lister 2006 rate mayor que 4
204 SELECT film_id AS "id película", title AS "Título del film", release_year AS "Año de estreno", rental_rate AS "Calificación"
205 FROM film
206 WHERE release_year = 2006
207 AND rental_rate > 4;
```

Data Output Messages Notifications

|   | Id película | Título del film   | Año de estreno | Calificación |
|---|-------------|-------------------|----------------|--------------|
| 1 | 133         | Chamber Italian   | 2006           | 4.99         |
| 2 | 384         | Grosse Wonderful  | 2006           | 4.99         |
| 3 | 8           | Airport Pollock   | 2006           | 4.99         |
| 4 | 98          | Bright Encounters | 2006           | 4.99         |

Total rows: 336 of 336    Query complete 00:00:00.123    Ln 203, Col 32

- **Function: f\_listar\_por\_anno\_y\_rate()**

Siguiendo el enfoque metodológico, agregaremos complejidad mediante una solución más elaborada, que permita recibir como parámetros el año y la calificación, para que la consulta sea dinámica.

```
```sql
CREATE OR REPLACE FUNCTION f_listar_por_anno_y_rate(
    _release_year INTEGER,
    _rental_rate NUMERIC
)
RETURNS TABLE (
    "Código de la Película" INTEGER,
    "Año de lanzamiento" INTEGER,
    "Título de la Película" character varying(255),
    "Calificación" NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT film_id AS "Código de la Película",
           release_year::INTEGER AS "Año de lanzamiento",

```

```

        title AS "Título de la Película",
        rental_rate AS "Calificación"
    FROM film
    WHERE release_year = _release_year AND rental_rate > _rental_rate
    ORDER BY title, rental_rate;
END;
$$;
```

```

- **Eliminar la función**

```

```sql
DROP FUNCTION IF EXISTS f_listar_por_anno_y_rate(integer, numeric);
```

```

- **Vista de la consulta usando la función "f\_listar\_por\_anno\_y\_rate()"**

287    **SELECT \* FROM f\_listar\_por\_anno\_y\_rate(2006, 4.0)**

|    | Código de la Película<br>integer | Año de lanzamiento<br>integer | Título de la Película<br>character varying | Calificación<br>numeric |
|----|----------------------------------|-------------------------------|--|-------------------------|
| 5  |                                  | 15                            | 2000                                       | All Forever             |
| 6  |                                  | 20                            | 2006                                       | Amelie Hellfighters     |
| 7  |                                  | 21                            | 2006                                       | American Circus         |
| 8  |                                  | 28                            | 2006                                       | Anthem Luke             |
| 9  |                                  | 31                            | 2006                                       | Apache Divine           |
| 10 |                                  | 32                            | 2006                                       | Apocalypse Flamingos    |
| 11 |                                  | 44                            | 2006                                       | Attacks Hate            |
| 12 |                                  | 45                            | 2006                                       | Attraction Newton       |
| 13 |                                  | 46                            | 2006                                       | Autumn Crow             |
| 14 |                                  | 47                            | 2006                                       | Baby Hall               |
| 15 |                                  | 48                            | 2006                                       | Backlash Undefeated     |
| 16 |                                  | 60                            | 2006                                       | Beast Hunchback         |
| 17 |                                  | 61                            | 2006                                       | Beauty Grease           |
| 18 |                                  | 65                            | 2006                                       | Behavior Runaway        |
| 19 |                                  | 68                            | 2006                                       | Betrayed Rear           |
| 20 |                                  | 70                            | 2006                                       | Bikini Borrowers        |
| 21 |                                  | 71                            | 2006                                       | Bilko Anonymous         |
| 22 |                                  | 74                            | 2006                                       | Birch Antitrust         |

Total rows: 336 of 336    Query complete 00:00:00.142    Ln 231, Col 68

Se puede verificar que sólo 2006 tiene datos mirando la tabla o, con una consulta que omita el año "2006", para que los liste todos:

```

289  SELECT film_id AS "id película", title AS "Título del film", release_year "Año de es·
290  FROM film
291  WHERE rental_rate > 4;
292

```

Data Output Messages Notifications SQL

|     | id película<br>integer | Título del film<br>character varying (255) | Año de estreno<br>integer | Calificación<br>numeric (4,2) |
|-----|------------------------|--|---------------------------|-------------------------------|
| 330 | 979                    | Witches Panic                              | 2006                      | 4.99                          |
| 331 | 980                    | Wizard Coldblooded                         | 2006                      | 4.99                          |
| 332 | 985                    | Wonderland Christmas                       | 2006                      | 4.99                          |
| 333 | 989                    | Working Microcosmos                        | 2006                      | 4.99                          |
| 334 | 994                    | Wyoming Storm                              | 2006                      | 4.99                          |
| 335 | 995                    | Yentl Idaho                                | 2006                      | 4.99                          |
| 336 | 1000                   | Zorro Ark                                  | 2006                      | 4.99                          |

Total rows: 336 of 336    Query complete 00:00:00.082    Ln 291, Col 24

## DICCIONARIO DE DATOS

En la documentación, se ha incluido el diccionario en formato csv, con el nombre "diccionario\_datos\_dvdrental.csv".

- **Script de creación del diccionario**

```

```sql
SELECT
    t1.TABLE_NAME AS "Tabla",
    t1.COLUMN_NAME AS "Columna",
    t1.COLUMN_DEFAULT AS "Valor por defecto",
    t1.IS_NULLABLE AS "Acepta nulos",
    t1.DATA_TYPE AS "Tipo de dato",
    COALESCE(t1.NUMERIC_PRECISION,
    t1.CHARACTER_MAXIMUM_LENGTH) AS "Longitud de columna",
    PG_CATALOG.COL_DESCRIPTION(t2.OID,
    t1.DTD_IDENTIFIER::int) AS "Descripción columna",
    t1.DOMAIN_NAME AS "Nombre dominio"
FROM
    INFORMATION_SCHEMA.COLUMNS t1
    INNER JOIN PG_CLASS t2 ON (t2.RELNAME = t1.TABLE_NAME)
WHERE
    t1.TABLE_SCHEMA = 'public'
ORDER BY
    t1.TABLE_NAME;
```

```

- **Vista de la consulta "Diccionario"**

```

309 v SELECT
310   t1.TABLE_NAME AS "Tabla",
311   t1.COLUMN_NAME AS "Columna",
312   t1.COLUMN_DEFAULT AS "Valor por defecto",
313   t1.IS_NULLABLE AS "Acepta nulos",
314   t1.DATA_TYPE AS "Tipo de dato",
315   COALESCE(t1.NUMERIC_PRECISION,
316             t1.CHARACTER_MAXIMUM_LENGTH) AS "Longitud de columna",
317   PG_CATALOG.COL_DESCRIPTION(t2.OID,
318                             t1.DTD_IDENTIFIER::int) AS "Descripción columna",
319   t1.DOMAIN_NAME AS "Nombre dominio"
320
321 FROM
322   INFORMATION_SCHEMA.COLUMNS t1
323   INNER JOIN PG_CLASS t2 ON (t2.RELNAME = t1.TABLE_NAME)
324 WHERE
325   t1.TABLE_SCHEMA = 'public'
326 ORDER BY
327   t1.TABLE_NAME;
  
```

Data Output Messages Notifications

SQL

| Tabla name | Columna name     | Valor por defecto | Acepta nulos | Tipo de dato                | Longitud de columna | Descripción |
|------------|------------------|-------------------|--------------|-----------------------------|---------------------|-------------|
| 127 store  | manager_staff_id | [null]            | NO           | smallint                    | 16                  | [null]      |
| 128 store  | last_update      | now()             | NO           | timestamp without time zone |                     | [null]      |

Total rows: 128 of 128    Query complete 00:00:00.118    Ln 324, Col 31

## PROCEDIMIENTO BACKUP BBDD

- **Consola**

Usando la línea de comandos de CMD o PS, se puede ingresar la siguiente instrucción de pg\_dump.

```

```
bash
pg_dump -h localhost -U evaluador -F t -f
"C:\backup_pgsql\backup_dvdrentalTest_JCordova.tar" dvdrentalTest
```
  
```

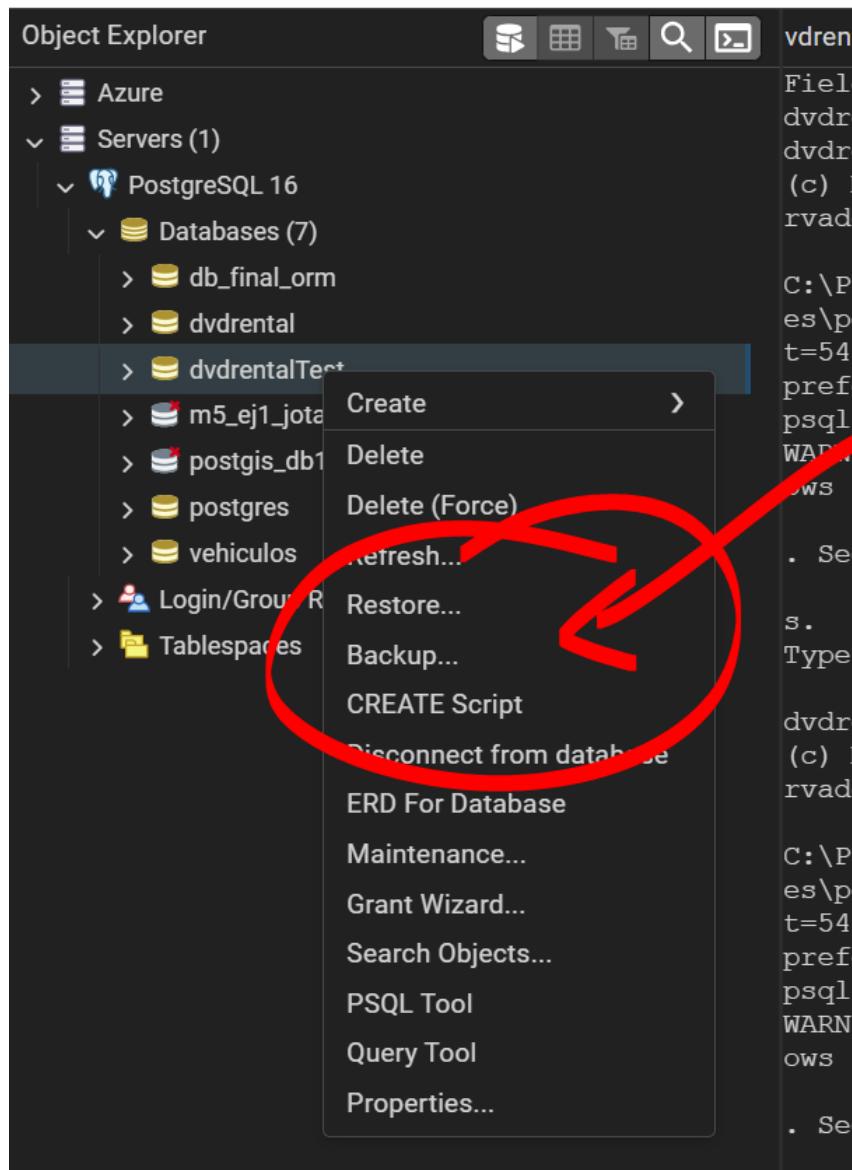
Significado de las opciones usadas:

- **-h localhost:** Especifica el host, en este caso "localhost".
- **-U evaluador:** El nombre de usuario que se utilizará para conectarse a la base de datos. Se creó un superuser "evaluador"
- **-F t:** Formato de salida "tar".
- **-f "C:\backup\_pgsql\backup\_dvdrentalTest\_JCordova.tar":** Ruta y nombre del archivo de respaldo.
- **"dvdrentalTest":** Nombre de la BBDD a respaldar.

- **PgAdmin**

Paso 1: Abrir la interfaz de PgAdmin y posicionar el cursor sobre la bbdd que se va a respaldar.

Paso 2: Hacer click sobre el nombre de la base y se desplegará un menú contextual, buscar la opción **"Backup"** y presionar.

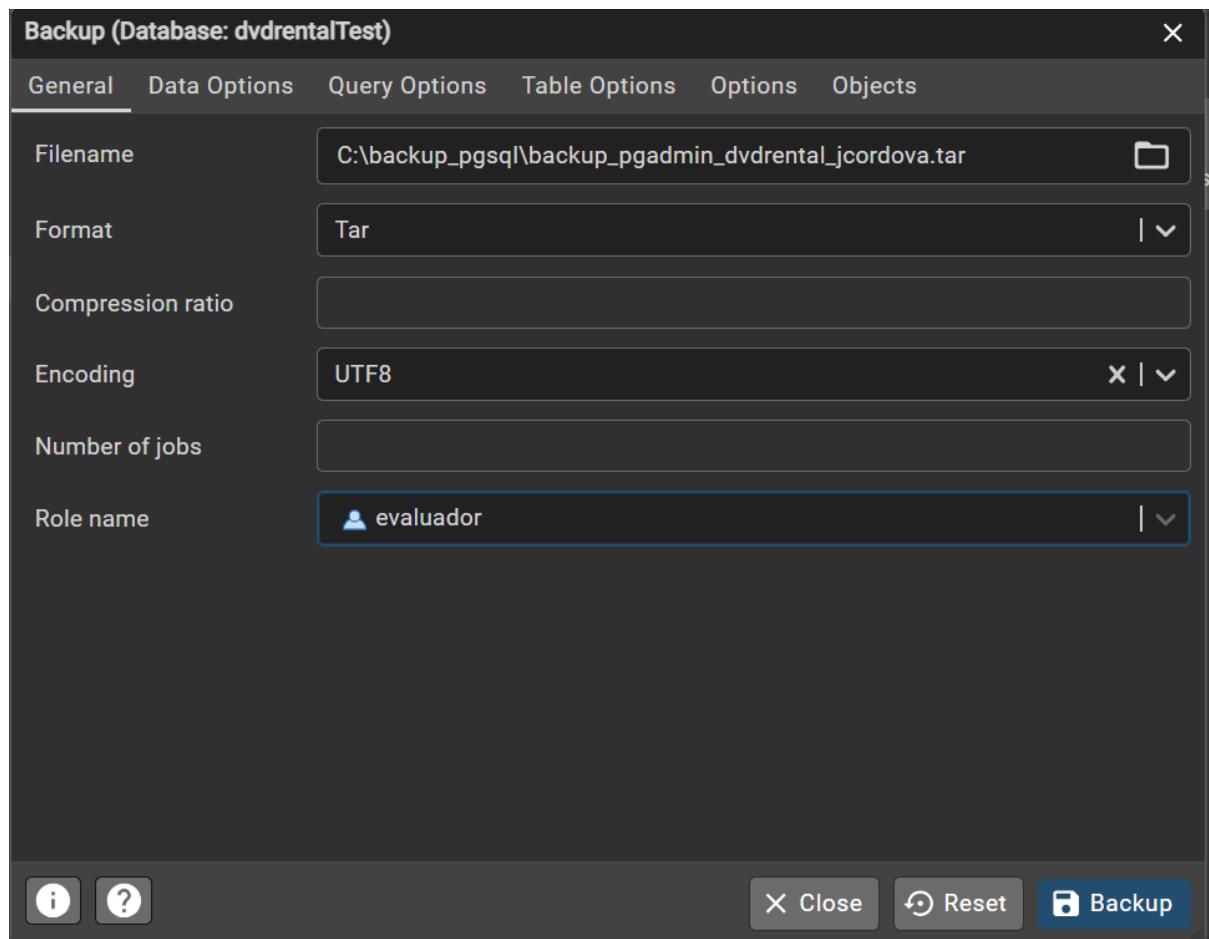


Paso 3: Se abrirá una ventana donde deberá llenar los campos:

- Filename: Un nombre identificatorio para el respaldo, en mi caso, "backup\_pgadmin\_dvdrental\_jcordova.tar", procurando que quede en la ruta donde se guardará, en este caso y como ejemplo.

"C:\backup\_pgsql\backup\_pgadmin\_dvdrental\_jcordova.tar"

- Format: Para este caso seleccionaremos "tar", formato comprimido.
- Compress ratio: Lo dejaremos en blanco
- Encoding: De la lista se escogerá UTF-8
- Number of jobs: No se toca.
- Role name: Indicamos que usuario tiene privilegios sobre la base. Para este caso, hemos creado un usuario distinto, para asegurar la trazabilidad e independencia de las acciones que pueda realizar, por lo tanto, se escoge "evaluador".



Paso 4: Presionar Aceptar y comenzará el proceso, al terminar sale un mensaje:

Revisar en el explorador de archivos

| Nombre                               | Fecha de modificación | Tipo         | Tamaño   |
|--------------------------------------|-----------------------|--------------|----------|
| backup_pgadmin_dvrental_jcordova.tar | 11/14/2024 4:30 PM    | tar Archive  | 2,807 KB |
| backup_pg.sh                         | 11/14/2024 3:55 PM    | Shell Script | 1 KB     |
| backup_dvrentalTest_JCordova.tar     | 11/14/2024 3:51 PM    | tar Archive  | 2,807 KB |
| dvrentalTest.tar                     | 10/15/2024 12:29 PM   | tar Archive  | 0 KB     |
| prueba_backup2_dvrentalTest.tar      | 10/9/2024 9:22 PM     | tar Archive  | 2,789 KB |
| test_backup_091024.tar               | 10/9/2024 9:07 PM     | tar Archive  | 2,789 KB |

## ANEXOS

- Superusuario

user: "evaluador"

pass: 123456

- **Archivos complementarios**

- SQL: evalFinalM5.sql
- Diccionario de datos: diccionario\_datos\_dvdrental.csv
- README: evalFinalM5\_Jorge\_Cordova.md
- HTML: evalFinalM5.html
- PDF: evalFinalM5.pdf
- Backup consola: backup\_dvdrentalTest\_JCordova.tar
- Backup PgAdmin: backup\_pgadmin\_dvdrental\_jcordova.tar