

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO  
FACULTAD DE INGENIERÍA  
ESCUELA DE INGENIERÍA INFORMÁTICA

## **TALLER DE BASES DE DATOS**

**JEAN RODRÍGUEZ**

INFORME DE TALLER VII

DICIEMBRE, 2019

# Índice

1. Introducción	1
2. Resolución de ejercicios	2
3. Conclusiones	10

## Lista de Figuras

1.	Segundo ejercicio - antes . . . . .	3
2.	Segundo ejercicio - después . . . . .	4
3.	Tercer ejercicio - antes . . . . .	5
4.	Tercer ejercicio - después . . . . .	5
5.	Cuarto ejercicio - antes . . . . .	6
6.	Restricciones en cascada . . . . .	7
7.	Cuarto ejercicio - después . . . . .	7
8.	Cuarto ejercicio - confirmación . . . . .	8
9.	Quinto ejercicio . . . . .	9

## 1. Introducción

Los modelos de negocio actuales manejan cantidades enormes de datos de los clientes en cada empresa, sea grande o pequeña, lo que evidencia una clara necesidad de un software capaz de procesar las solicitudes requeridas. Cuando una entidad desea minimizar la complejidad de sus operaciones y mantener tiempos de respuesta aceptables, se debe dejar en claro un diseño simple y eficiente que sea mantenible y facilite la realización de cambios a futuro.

Con la intención de demostrar un correcto funcionamiento en la base de datos propuesta, se deja a disposición una serie de consultas para el motor PostgreSQL que entregan respuestas de manera rápida y efectiva, utilizando como base el script inicial para cargar la base de datos, y su correspondiente archivo de valores separado por comas. Además, se deja a disposición una copia de la base de datos posterior a las modificaciones realizadas luego de realizar los ejercicios propuestos, que se encuentra en formato SQL plano para cargar mediante una query.

## 2. Resolución de ejercicios

A continuación se presenta el conjunto de preguntas y respuestas correspondientes en el lenguaje PostgreSQL con los accesos necesarios a las tablas y sus llaves, y una introducción a lo que sería plpgsql.

1. Crear un manual paso a paso como se ejecuta el lenguaje plpgsql en postgresql.

I. Se realiza una delimitación, esto quiere decir que lo que exista dentro de los límites asignados se ejecutará como una serie de instrucciones que no están especificadas en el lenguaje estándar de PostgreSQL. En Postgre, generalmente suele utilizarse un doble signo de dólar (\$\$). Previo a esto se puede especificar el procedimiento a realizar en un bloque externo (DO, CREATE FUNCTION, etc)

II. Al interior de la delimitación se ponen instrucciones válidas para plpgsql dentro de un BEGIN y END, estas incluyen:

- Iteraciones en loop
- Cursores
- Disparadores de funciones propias
- Tipos de datos complejos como retorno de funciones (registros, conjuntos, tablas)
- Entre otros

Para más información, visitar la tabla de contenidos en el sitio oficial:

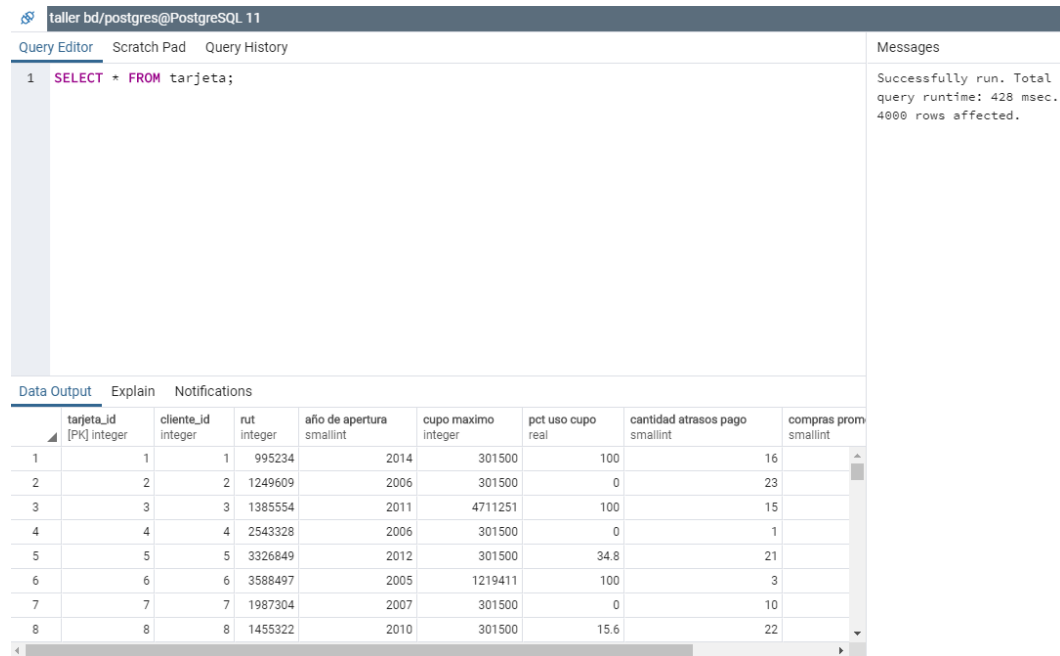
<https://www.postgresql.org/docs/10/plpgsql.html>

III. Luego de cerrar la delimitación, se especifica el lenguaje con la sentencia:

```
LANGUAGE plpgsql;
```

2. Actualizar el porcentaje de uso de cupos en 10% de los productos.

Primero, se realiza una consulta para confirmar el porcentaje de uso de cupo de algunos individuos, con el fin de mostrar el correcto funcionamiento del script a ejecutar.



The screenshot shows a PostgreSQL query editor interface. At the top, the title bar reads "taller bd/postgres@PostgreSQL 11". Below it, there are tabs for "Query Editor", "Scratch Pad", and "Query History". The "Query Editor" tab is active, displaying a single SQL query: "1 SELECT \* FROM tarjeta;". To the right of the query editor, a "Messages" panel shows a confirmation message: "Successfully run. Total query runtime: 428 msec. 4000 rows affected." Below the query editor, there are tabs for "Data Output", "Explain", and "Notifications". The "Data Output" tab is selected, displaying a table with 8 rows and 9 columns. The columns are: "tarjeta\_id [PK] integer", "cliente\_id integer", "rut integer", "año de apertura smallint", "cupo maximo integer", "pct uso cupo real", "cantidad atrasos pago smallint", and "compras prom smallint". The table contains data for 8 different cards, with values ranging from 1 to 8 for the first two columns, and various values for the remaining columns.

	tarjeta_id [PK] integer	cliente_id integer	rut integer	año de apertura smallint	cupo maximo integer	pct uso cupo real	cantidad atrasos pago smallint	compras prom smallint
1	1	1	995234	2014	301500	100	16	
2	2	2	1249609	2006	301500	0	23	
3	3	3	1385554	2011	4711251	100	15	
4	4	4	2543328	2006	301500	0	1	
5	5	5	3326849	2012	301500	34.8	21	
6	6	6	3588497	2005	1219411	100	3	
7	7	7	1987304	2007	301500	0	10	
8	8	8	1455322	2010	301500	15.6	22	

Figura 1: Segundo ejercicio - antes

La figura 2 muestra como se modifica correctamente el campo de cada cliente, considerando que al agregar el 10% extra a cada tarjeta se deben respetar el limite mínimo y máximo (0 y 100 respectivamente), por lo que los casos de modificación varían según la suma del argumento ingresado y el porcentaje de uso mismo. Además, se observa que la función toma como primer parámetro un arreglo de tarjetas, para así modificar solo aquellas necesarias, en este caso todas las tarjetas se pueden obtener con la query del primer argumento.

taller bd/postgres@PostgreSQL 11

Query Editor

Scratch Pad

Query History

1

CREATE OR REPLACE FUNCTION agregar\_pct\_uso\_cupo(arreglo integer[], pct real) RETURNS void AS \$\$

2

BEGIN

3

UPDATE tarjeta

4

SET "pct uso cupo" = (CASE WHEN tarjeta.tarjeta\_id = ANY(arreglo) THEN

5

CASE

6

WHEN "pct uso cupo" + pct BETWEEN 0 AND 100 THEN "pct uso cupo" + pct

7

WHEN "pct uso cupo" + pct < 0 THEN 0

8

WHEN "pct uso cupo" + pct > 100 THEN 100

9

END

10

END);

11

END

12

\$\$ LANGUAGE plpgsql;

13

SELECT agregar\_pct\_uso\_cupo((select array\_agg(tarjeta\_id) from tarjeta), 10);

14

SELECT \* FROM tarjeta;

15

Messages

Successfully run. Total query runtime: 346 msec. 4000 rows affected.

Data Output

Explain

Notifications

	tarjeta_id [PK] integer	cliente_id integer	rut integer	año de apertura smallint	cupo maximo integer	pct uso cupo real	cantidad atrasos pago smallint	compras promedio smallint
1		1	1	995234	2014	301500	100	16
2		2	2	1249609	2006	301500	10	23
3		3	3	1385554	2011	4711251	100	15
4		4	4	2543328	2006	301500	10	1
5		5	5	3326849	2012	301500	44.8	21
6		6	6	3588497	2005	1219411	100	3
7		7	7	1987304	2007	301500	10	10
8		8	8	1455322	2010	301500	25.6	22

Figura 2: Segundo ejercicio - después

### 3. Actualizar el Estado civil del cliente cuyo nombre es Maria Cristina Cedres a soltero

Primero se ejecuta una query para mostrar los datos a modificar y luego ejecutamos la instrucción para cambiar el estado civil de Maria, mostrando así un antes y un después.



The screenshot shows the PostgreSQL Query Editor interface. The query editor contains the following SQL query:

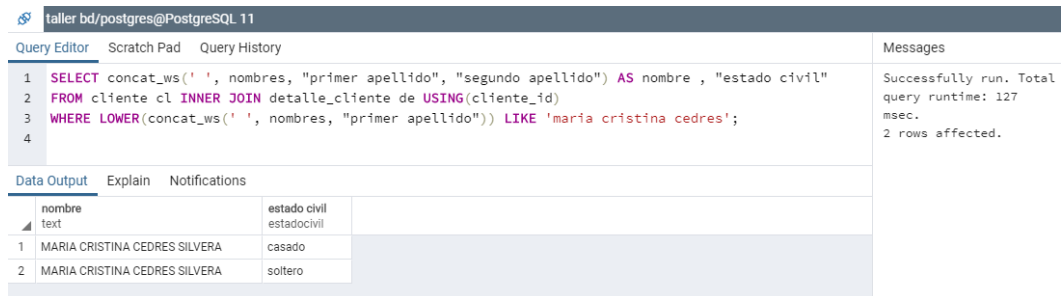
```
1 SELECT concat_ws(' ', nombres, "primer apellido", "segundo apellido") AS nombre, "estado civil"
2 FROM cliente cl INNER JOIN detalle_cliente de USING(cliente_id)
3 WHERE LOWER(concat_ws(' ', nombres, "primer apellido")) LIKE 'maria cristina cedres';
4
```

The Messages pane on the right indicates: "Successfully run. Total query runtime: 127 msec. 2 rows affected."

The Data Output pane shows the following results:

	nombre text	estado civil estadocivil
1	MARIA CRISTINA CEDRES SILVERA	casado
2	MARIA CRISTINA CEDRES SILVERA	soltero

Figura 3: Tercer ejercicio - antes



The screenshot shows the PostgreSQL Query Editor interface. The query editor contains the following SQL query:

```
1 SELECT concat_ws(' ', nombres, "primer apellido", "segundo apellido") AS nombre, "estado civil"
2 FROM cliente cl INNER JOIN detalle_cliente de USING(cliente_id)
3 WHERE LOWER(concat_ws(' ', nombres, "primer apellido")) LIKE 'maria cristina cedres';
4
```

The Messages pane on the right indicates: "Successfully run. Total query runtime: 127 msec. 2 rows affected."

The Data Output pane shows the following results:

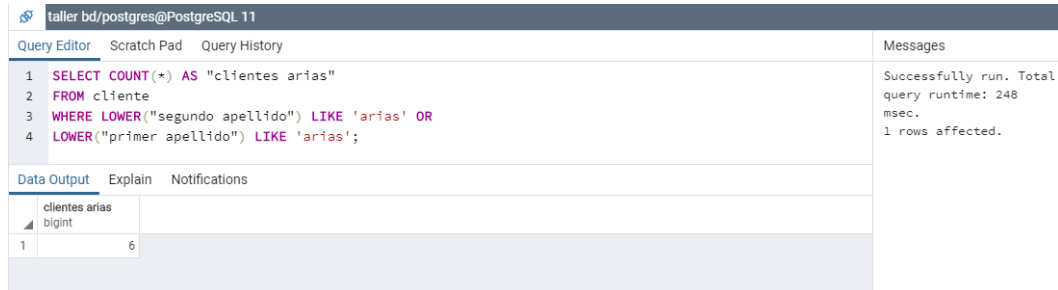
	nombre text	estado civil estadocivil
1	MARIA CRISTINA CEDRES SILVERA	casado
2	MARIA CRISTINA CEDRES SILVERA	soltero

Figura 4: Tercer ejercicio - después



#### 4. Borrar al cliente Arias.

Se comienza mostrando los datos previos a la modificación, para asegurar su correcto funcionamiento.




The screenshot shows a PostgreSQL query editor interface. The title bar indicates the user is 'taller bd/postgres@PostgreSQL 11'. The interface is divided into several sections: 'Query Editor', 'Scratch Pad', 'Query History', and 'Messages'. The 'Query Editor' contains a SQL query with four lines: 1. `SELECT COUNT(*) AS "clientes arias"`, 2. `FROM cliente`, 3. `WHERE LOWER("segundo apellido") LIKE 'arias' OR`, and 4. `LOWER("primer apellido") LIKE 'arias';`. The 'Messages' section on the right shows a successful execution: 'Successfully run. Total query runtime: 248 msec. 1 rows affected.' Below the query editor, there is a 'Data Output' section with tabs for 'Data Output', 'Explain', and 'Notifications'. The 'Data Output' tab is active, showing a table with one column 'clientes arias' of type 'bigint'. The table contains one row with the value '6'.

clientes arias
6

Figura 5: Cuarto ejercicio - antes

Además, se modifican las restricciones de alteración en las tablas que poseen claves foráneas, para que no existan datos sobrantes al eliminar al cliente (ver figura 6).

Luego de ejecutar la query para eliminar al cliente, se demuestra que no existan datos sobrantes relacionados con este, y ya que se sabe con antelación que el total de tarjetas es de 4000, solo basta con saber que al eliminar 6 clientes de apellido Arias quedara un total de 3994 tarjetas.


 taller bd/postgres@PostgreSQL 11
 

Query Editor
 Scratch Pad
 Query History

```

1 ALTER TABLE detalle_cliente
2 DROP CONSTRAINT detalle_cliente_cliente_id_fkey,
3 ADD CONSTRAINT detalle_cliente_cliente_id_fkey FOREIGN KEY (cliente_id)
4 REFERENCES cliente(cliente_id) ON UPDATE CASCADE ON DELETE CASCADE;
5
6 ALTER TABLE tarjeta
7 DROP CONSTRAINT tarjeta_cliente_id_fkey,
8 ADD CONSTRAINT tarjeta_cliente_id_fkey FOREIGN KEY (cliente_id)
9 REFERENCES cliente(cliente_id) ON UPDATE CASCADE ON DELETE CASCADE;
10
11 ALTER TABLE compra
12 DROP CONSTRAINT compra_tarjeta_id_fkey,
13 ADD CONSTRAINT compra_tarjeta_id_fkey FOREIGN KEY (tarjeta_id)
14 REFERENCES tarjeta(tarjeta_id) ON UPDATE CASCADE ON DELETE CASCADE;
15
16 ALTER TABLE producto
17 DROP CONSTRAINT producto_compra_id_fkey,
18 ADD CONSTRAINT producto_compra_id_fkey FOREIGN KEY (compra_id)
19 REFERENCES compra(compra_id) ON UPDATE CASCADE ON DELETE CASCADE;
  
```

Figura 6: Restricciones en cascada

 taller bd/postgres@PostgreSQL 11
 

Query Editor
 Scratch Pad
 Query History

```

1 SELECT COUNT(*) AS "clientes arias"
2 FROM cliente
3 WHERE LOWER("segundo apellido") LIKE 'arias' OR
4 LOWER("primer apellido") LIKE 'arias';
5
  
```

Data Output
 Explain
 Notifications

	clientes arias
	bigint
1	0

Messages
 

Successfully run. Total query runtime: 116 msec.  
 1 rows affected.

Figura 7: Cuarto ejercicio - después

taller bd/postgres@PostgreSQL 11

Query Editor Scratch Pad Query History

1 `SELECT COUNT(*) FROM tarjeta;`

Data Output Explain Notifications

	count
	bigint
1	3994

Messages

Successfully run. Total query runtime: 106 msec.  
1 rows affected.

Figura 8: Cuarto ejercicio - confirmación

5. Escriba un procedimiento que reciba como parámetro un nombre de un cliente y que devuelva el número de veces que compra en promedio al año.

taller bd/postgres@PostgreSQL 11

Query Editor Scratch Pad Query History Messages

```

1 CREATE OR REPLACE FUNCTION compras_promedio(nombre_cliente TEXT) RETURNS
2 TABLE("id de cliente" integer, "id de tarjeta" integer, "compras promedio del cliente" smallint) AS $$
3 BEGIN
4     RETURN QUERY
5     SELECT cliente_id, tarjeta_id, "compras promedio"
6     FROM cliente INNER JOIN tarjeta USING(cliente_id)
7     WHERE LOWER(concat_ws(' ', nombres, "primer apellido", "segundo apellido")) LIKE LOWER(nombre_cliente);
8 END
9 $$ LANGUAGE plpgsql;
10
11 SELECT * FROM compras_promedio('alberto oscar abalos rochon');

```

Successfully run. Total query runtime: 222 msec. 2 rows affected.

Data Output Explain Notifications

	id de cliente integer	id de tarjeta integer	compras promedio del cliente smallint
1	4	4	6
2	2672	2672	7

Figura 9: Quinto ejercicio

### **3. Conclusiones**

Del presente informe se obtuvo en detalle el proceso de organización necesario para conformar la base de datos propuesta para el cliente ficticio, de tal manera que facilita la construcción del software que debe utilizar las variables necesarias para que el negocio funcione de manera exitosa, dejándose expuesta una interpretación clara del modelo definido para entregar tiempos de respuesta breves en futuras consultas.

Además, se obtuvo resultados satisfactorios a la hora de aplicar en su conjunto el conocimiento aplicado en las entregas previas, realización exitosa de modificaciones a la base de datos y la incorporación de nueva materia relacionada al lenguaje procedimental plpgsql.