

# SQL

## Postgres

### Tipos de datos

Para poder definir el esquema de una tabla, es necesario especificar el tipo de dato de cada uno de sus campos. En postgres los tipos de datos más comunes son:

- Números
- Cadenas de caracteres
- Fechas y horas

El resto de los tipos se pueden encontrar en la documentación oficial

- <https://www.postgresql.org/docs/current/datatype.html>

Para propósitos prácticos se revisarán los 3 tipos indicados en el documento:

#### Números

Los valores numéricos los podemos clasificar en:

- **Enteros** no cuentan con parte decimal como: smallint, integer, bigint
- **Flotantes**, aquellos que cuentan con una parte decimal: decimal, numeric, double precision.

Name	Storage Size	Description	Range
smallint	2 bytes	small-range Integer	-32768 to +32767
integer	4 bytes	typical choice for Integer	-2147483648 to +2147483647
bigint	8 bytes	large-range Integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing Integer	1 to 32767
serial	4 bytes	autoincrementing Integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing Integer	1 to 9223372036854775807

#### Cadenas de caracteres

Permiten almacenar texto, representado en cadenas de caracteres de tamaño N (longitud).

Name	Description
<code>character varying(<i>n</i>), varchar(<i>n</i>)</code>	variable-length with limit
<code>character(<i>n</i>), char(<i>n</i>)</code>	fixed-length, blank padded
<code>text</code>	variable unlimited length

## Fechas

Permiten almacenar fechas y tiempo. Una de las principales ventajas de usar este formato es que nos permite realizar operaciones con fechas.

Algunos ejemplos de operaciones son:

- Restar N número de días a una fecha en particular
- Extraer el mes de una fecha
- Restar un intervalo de tiempo a una fecha

Operaciones en postgres: <https://www.postgresql.org/docs/current/functions-datetime.html>

Name	Storage Size	Description	Low Value	High Value	Resolution
<code>timestamp [ (<i>p</i>) ] [ without time zone ]</code>	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
<code>timestamp [ (<i>p</i>) ] with time zone</code>	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
<code>date</code>	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
<code>time [ (<i>p</i>) ] [ without time zone ]</code>	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
<code>time [ (<i>p</i>) ] with time zone</code>	12 bytes	time of day (no date), with time zone	00:00:00+1559	24:00:00-1559	1 microsecond
<code>interval [ <i>fields</i> ] [ (<i>p</i>) ]</code>	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

## Instrucciones básicas del Lenguaje de Definición de Datos (LDD)

### Crear Database

```
create database <nombre_base>;
```

### Crear schema

- Los esquemas (schemas) nos ayudan a separar de manera lógica las tablas.

Sintaxis:

```
CREATE SCHEMA schema_name;
```

## Crear tablas

Sintaxis:

```
CREATE TABLE nombre_esquema.table_name (  
    column1 datatype(length),  
    column2 datatype(length),  
    column3 datatype(length)  
);
```

Nota:

- Observar que la definición de los campos se separa por coma con excepción del último.

Crear tablas a partir de otra:

```
CREATE TABLE schema_name.new_table as select * from schema_name.original_table;
```

## Eliminar tablas

```
DROP TABLE [IF EXISTS] table_name [CASCADE | RESTRICT];
```

## Modificar estructura de una tabla

- Agregar campos

```
ALTER TABLE <table_name> ADD COLUMN <column_name> <datatype>;
```

Nota: los campos siempre se agregan al “final”.

- Quitar campos

```
ALTER TABLE <table_name> DROP COLUMN <column_name>;
```

- Agregar restricciones de llave foránea

# Instrucciones DML

## Agregar / Modificar contenido de una tabla

- Insertar registros

```
INSERT INTO table_name(column1, column2, ...)  
VALUES (value1, value2, ...);
```

- Cargar datos de un archivo en una tabla: **COPY**

El comando copy permite cargar un archivo, generalmente, csv o texto a una tabla previamente creada.

Se requieren:

1. Ruta del archivo. Se debe contar con acceso de lectura en dónde se encuentra el archivo.
2. Tabla destino, que deberá estar creada antes de la carga.
3. Definición de un separador.
4. Indicar si el archivo cuenta con un encabezado.

Sintaxis:

```
COPY tabla(column_a, column_b, ... , column)  
FROM 'RUTA\file.csv'  
DELIMITER ','  
CSV HEADER;
```

Ejemplo:

### Import a CSV file into a table using COPY statement

To import this CSV file into the **persons** table, you use **COPY** statement as follows:

```
COPY persons(first_name, last_name, dob, email)  
FROM 'C:\sampledb\persons.csv'  
DELIMITER ','  
CSV HEADER;
```

Versión corta:

- Si el 'layout' del archivo coincide con el orden de las columnas de la tabla, podríamos sólo indicar el nombre de la tabla:

```
COPY tabla
FROM 'RUTA\file.csv'
DELIMITER ','
CSV HEADER;
```

Referencia:

<https://www.postgresqltutorial.com/postgresql-tutorial/import-csv-file-into-posgresql-table/>

- Editar el contenido de una tabla

-- **Importante:** esto afecta a toda la tabla.

```
UPDATE table_name set colum_name = new_value;
```

-- Si sólo se desea modificar registros que cumplan con una condición.

```
UPDATE table_name set colum_name = new_value WHERE cond = 1;
```

- Eliminar contenido de una tabla:

-- Importante: borra todos los registros de la tabla

```
DELETE FROM table_name;
```

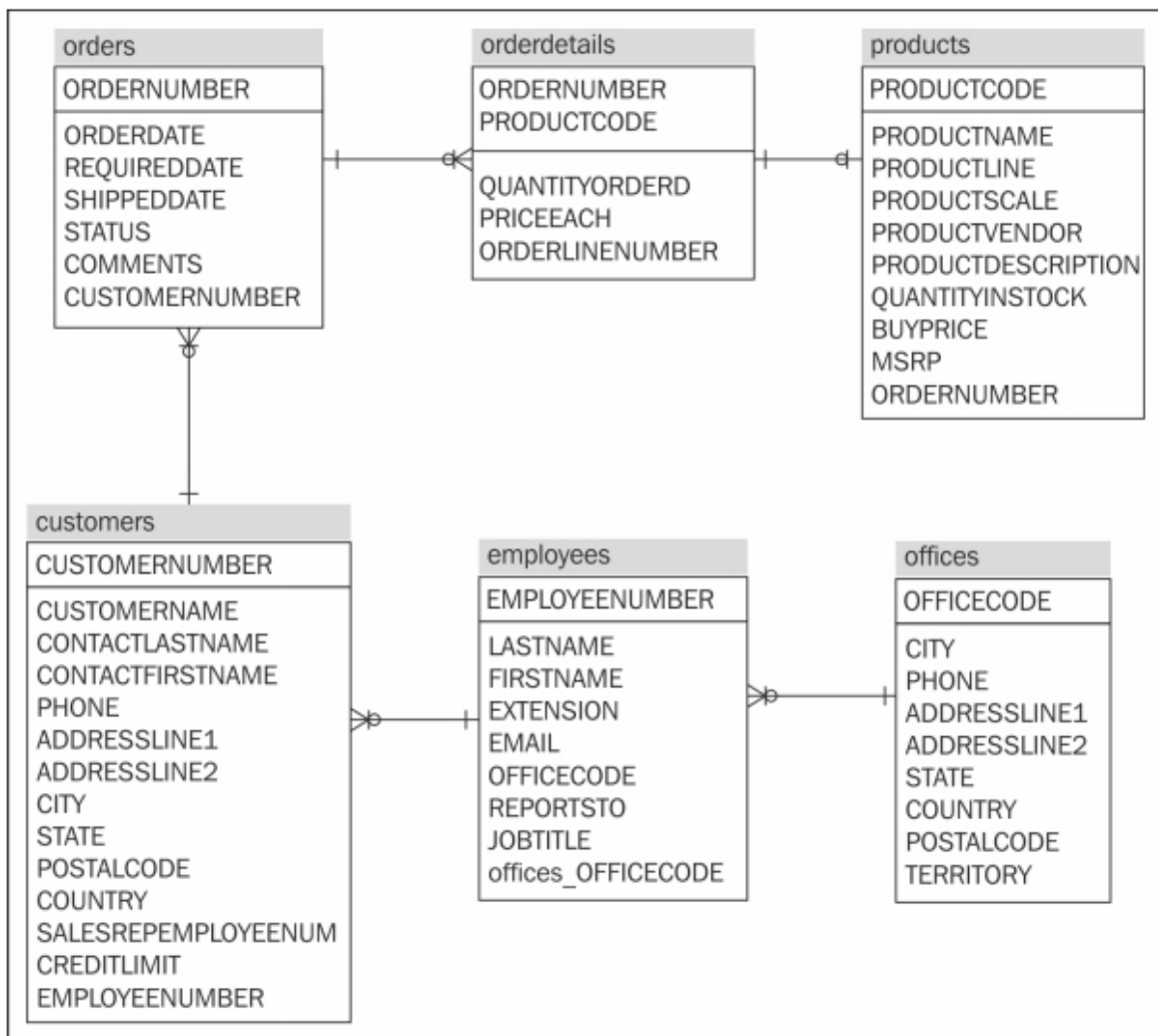
```
DELETE FROM table_name where colum = value;
```

-- Borra todos los registros de la tabla pero es mucho más eficiente

```
TRUNCATE table_name;
```

## Hands-on: DDL y DML

Para practicar los comandos se utilizará el caso de uso de steel-wheels que es una base de datos de ejemplo de una tienda de vehículos.



**Paso 1:** Conectar a la base de datos postgres

**Paso 2:** Ejecutar las siguientes sentencias de creación:

```
DROP TABLE IF EXISTS offices;
CREATE TABLE offices (
  OFFICECODE varchar(50) NOT NULL default "",
  CITY varchar(50) NOT NULL default "",
  PHONE varchar(50) NOT NULL default "",
  ADDRESSLINE1 varchar(50) NOT NULL default "",
  ADDRESSLINE2 varchar(50) default NULL,
  STATE varchar(50) default NULL,
  COUNTRY varchar(50) NOT NULL default "",
  POSTALCODE varchar(10) NOT NULL default "",
  TERRITORY varchar(10) NOT NULL default "",
  PRIMARY KEY(OFFICECODE)
);
```

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees (
  EMPLOYEENUMBER bigint NOT NULL default '0',
  LASTNAME varchar(50) NOT NULL default "",
  FIRSTNAME varchar(50) NOT NULL default "",
  EXTENSION varchar(10) NOT NULL default "",
  EMAIL varchar(100) NOT NULL default "",
  OFFICECODE varchar(20) NOT NULL default "",
  REPORTSTO bigint default NULL,
  JOBTITLE varchar(50) NOT NULL default "",
  PRIMARY KEY(EMPLOYEENUMBER)
);
```

```
DROP TABLE IF EXISTS customers;
CREATE TABLE customers (
  CUSTOMERNUMBER bigint NOT NULL default '0',
  CUSTOMERNAME varchar(50) NOT NULL default "",
  CONTACTLASTNAME varchar(50) NOT NULL default "",
  CONTACTFIRSTNAME varchar(50) NOT NULL default "",
  PHONE varchar(50) NOT NULL default "",
  ADDRESSLINE1 varchar(50) NOT NULL default "",
  ADDRESSLINE2 varchar(50) default NULL,
  CITY varchar(50) NOT NULL default "",
  STATE varchar(50) default NULL,
  POSTALCODE varchar(15) default NULL,
  COUNTRY varchar(50) NOT NULL default "",
  SALESREPEMPLYEENUMBER bigint default NULL,
  CREDITLIMIT decimal(17,0) default NULL,
  PRIMARY KEY(CUSTOMERNUMBER)
);
```

```
DROP TABLE IF EXISTS orders;
CREATE TABLE orders (
  ORDERNUMBER bigint NOT NULL default '0',
  ORDERDATE timestamp NOT NULL default CURRENT_TIMESTAMP,
  REQUIREDDATE timestamp NOT NULL default '0001-01-01 00:00:00',
  SHIPPEDDATE timestamp NOT NULL default '0001-01-01 00:00:00',
  STATUS varchar(15) NOT NULL default "",
  COMMENTS text,
```

```

CUSTOMERNUMBER bigint NOT NULL default '0',
PRIMARY KEY(ORDERNUMBER)
);

DROP TABLE IF EXISTS orderdetails;
CREATE TABLE orderdetails (
  ORDERNUMBER bigint NOT NULL default '0',
  PRODUCTCODE varchar(50) NOT NULL default "",
  QUANTITYORDERED bigint NOT NULL default '0',
  PRICEEACH decimal(17,0) NOT NULL default '0',
  ORDERLINENUMBER smallint NOT NULL default '0',
  PRIMARY KEY(ORDERNUMBER,PRODUCTCODE)
);

DROP TABLE IF EXISTS products;
CREATE TABLE products (
  PRODUCTCODE varchar(50) NOT NULL default "",
  PRODUCTNAME varchar(70) NOT NULL default "",
  PRODUCTLINE varchar(50) NOT NULL default "",
  PRODUCTSCALE varchar(10) NOT NULL default "",
  PRODUCTVENDOR varchar(50) NOT NULL default "",
  PRODUCTDESCRIPTION text NOT NULL,
  QUANTITYINSTOCK smallint NOT NULL default '0',
  BUYPRICE decimal(17,0) NOT NULL default '0',
  MSRP decimal(17,0) NOT NULL default '0',
  PRIMARY KEY(PRODUCTCODE)
);

```

**Paso 3:** Descargar los archivos y colocarlos en una ruta con permisos públicos.

- Windows:



- Linux:

- /tmp/

**Paso 4:** Cargar información en las tablas mediante el comando COPY

```

COPY offices FROM '/tmp/offices.csv' DELIMITER ',' CSV HEADER;

COPY employees FROM '/tmp/employees.csv' DELIMITER ',' CSV HEADER;

COPY customers FROM '/tmp/customers.csv' DELIMITER ',' CSV HEADER;

COPY orders FROM '/tmp/orders.csv' DELIMITER ',' CSV HEADER;

```



```
COPY orderdetails FROM '/tmp/orderdetails.csv' DELIMITER ',' CSV HEADER;
```

```
COPY products FROM '/tmp/products.csv' DELIMITER ',' CSV HEADER;
```

**Paso 5:** Realizar las siguientes tareas

a) Realizar una copia de la tabla *offices* y llamarla *offices\_copy* – Sandoval Cassani, Andrea Itzel

```
COPY offices FROM '/tmp/offices.csv' DELIMITER ',' CSV HEADER;
```

b) Agregar una columna a la nueva tabla: *offices\_copy* – Dante Romero Nava

```
ALTER TABLE offices_copy  
ADD columna_nueva varchar(255);
```

c) Editar un registro de la tabla *offices\_copy* – Lecona Valdespino, Carlos Natanael

```
Update offices_copy set phone = '+1 650 234 5674' where officecode = 1;
```

d) Borrar un registro de *offices\_copy* – Martínez Calzada, Susana Paola

```
DELETE FROM offices_copy where OFFICECODE=1;
```

e) Borrar todo el contenido de la tabla *offices\_copy* – Chirinos Funatsu, Jonatan Zeedxi

```
DELETE FROM offices_copy
```

f) Borrar la tabla *offices\_copy* – Báez Gómez Tagle, Enrique Ulises

```
DROP TABLE IF EXISTS offices_copy
```

## Instrucciones DQL (Parte 1: Álgebra relacional)

- proyección
- selección
- unión
- diferencia
- producto cartesiano
- join

## Hands-on: DQL (Álgebra relacional)

Obtener la siguiente información:

**Ejercicio 1:** Listar: Nombres de clientes, Nombre del representante de venta, Apellido del representante de venta.

Rodrigo Lucas Nieto

```
select cu.customername, e.firstname, e.lastname  
from customers cu  
join employees e on cu.salesrepemployeenumber = e.employeenumber;
```

**Ejercicio 2:** Listar, Nombres de clientes, Nombre del representante de venta, Apellido del representante de venta de clientes de 'Hong Kong'.

Chirinos Funatsu, Jonatan Zeedxi :/ :(

```
select cu.customername, e.firstname, e.lastname, cu.country  
from customers cu  
join employees e on cu.salesrepemployeenumber = e.employeenumber;  
where country like "%Hong Kong%";
```

**Ejercicio 3:** Listar, Nombres de clientes, Nombre del representante de venta, Apellido del representante de venta de clientes de 'Hong Kong' que tengan asociada alguna orden.

**Ejercicio 4:** Listar ordenes que tengan productos de la línea 'Motorcycles'

Campos a mostrar:

- ordernumber
- orderdate
- product name
- status

Enrique Ulises Baez Gómez Tagle

```
select o.ordernumber, o.orderdate, p.productname, o.status  
from orders o  
join orderdetails od on o.ordernumber = od.ordernumber  
join products p on od.productcode = p.productcode  
where p.productline = 'Motorcycles';
```

**Ejercicio 5:** Proponer el uso de la operación unión:

Iván Vega Matías

```
SELECT ORDERNUMBER, PRODUCTCODE from orders union SELECT
```

```
ORDERNUMBER, PRODUCTCODE from products;
```

**Ejercicio 6:** Proponer el uso de la operación producto cartesiano:

[Reinoso Fuentes, Lorenzo](#)

```
SELECT cu.customername, e.email  
FROM customers as cu, employees as e
```

## Instrucciones DQL (Parte 2)

- Ordenamiento
- Campo calculados
- Agrupación de datos
- Like / IN / NOT IN
- Case when
- Subconsultas

### ● Contar número de registros

```
select count(*) from <table_name>;
```

– Si se indica el nombre del campo en particular, si dicho campo es nulo se cuenta como 0.

```
select count(campos) from <table_name>;
```

### ● Ordenamiento

```
select * from <table> order by field;
```

-- Por defecto el orden es ascendente de menor a mayor.  
Existen dos formas: asc, desc

```
select * from <table> order by field desc;
```

```
select * from <table> order by field1 asc, field2 desc; --Podemos ordenar por varios  
campos, y cada uno con su propio criterio.
```

## ● Campos calculados

Existen una serie de operaciones aritméticas que ofrecen los manejadores

- count(\*) -- se menciona arriba
- sum(field)
- avg(field)
- max(field)
- min(field)

```
select
  count(*), sum(field), avg(field), max(field), min(field)
from <table_name>
```

## ● Agrupaciones + campos calculados

La sentencia group nos permite hacer agrupaciones de valores de campos.

```
select
  field1
from <table>
group by field1;
```

**Nota:** El listado de campos a seleccionar deben ser los mismos utilizados para la agrupación no marca error.

Correcto	Incorrecto
<pre>select   field1 from table group by field1;</pre>	<pre>select   field1,   field2 from table group by field1;</pre>
<pre>select   field1,   count(field1) from table group by field1;</pre>	<pre>select   field1,   field2,   count(field2) from table group by field1;</pre>

Ejemplo: En la tabla de lado derecho se muestra una lista de frutas, del lado derecho se muestran los grupos resultantes por el campo color. Serían 3 grupos por color:

- rojo
- naranja
- verde

Lo interesante que podemos hacer operaciones aritméticas con los elementos que conforman cada grupo, cómo:

- total de registros de frutas de color rojo, serían 3.
- total de registros de frutas de color verde, serían 2.

				grupo	total x grupo	sumatoria x grupo
id	fruta	color	precio	color	count()	sum()
1	fresa	rojo	2	rojo	3	6
2	sandía	rojo	2	naranja	1	6
3	frambruesa	rojo	2	verde	2	4
4	zanahoria	naranja	3			
5	espinacas	verde	3			
6	nopales	verde	4			

Este tipo de conteos y operaciones, nos sirve para contestar por ejemplo:

- ¿Cuántos usuarios por país existen?
- ¿Cuáles fueron el total de ventas por mes?

Group by también nos sirve para de-duplicar o obtener valores diferentes.

Por ejemplo:

```
select color from frutas
```

--Nos mostraría los 3 valores de colores en este caso. Lo que podemos interpretar como los valores diferentes que se encuentran registrados en la tabla.

## • Distinct

Permite obtener los valores diferentes del campo(s) indicados posterior a select.

Es útil para cuando se desea explorar los posibles valores de un campo.

```
select distinct field1, field2
from table;
```

--La sentencia group es otra forma de obtener distintos

```
select field1, field2
from table_name
group by field1, field2;
```

## • Búsquedas: LIKE, IN, NOT IN

- **LIKE**: Permite filtrar aquellos registros que cumplan con la(s) palabras(s) indicadas. La diferencia entre LIKE y la igualdad (=), radica en que Like permite el uso de comodines para la búsqueda lo que permite sea un poco más flexible.

```
select *
from <table_name>
where field like '%palabra%palabra%';
```

- **IN:** Retorna aquellos registros cuyos valores se encuentren en la lista proporcionada.

```
select
*
from <table_name>
where field in ('value1', 'value2', 'valueN');
```

- **NOT IN:** Retorna aquellos registros cuyos valores no se encuentren en la lista proporcionada.

```
select
*
from <table_name>
where field not in ('field1', 'field2');
```

## ● Case when


Es el equivalente a un IF/THEN de lenguajes de programación.

```
select
    case when condicion then valor else valor end as nombre
from tabla;
```

Ejemplo:

```
select
    id
    , case when gender = 'F' then 'Female' else 'Male' end as gender
from <table_name>;
```

id	gender
1	F
2	M
3	F




id	gender
1	Female
2	Male
3	Female

¿Qué pasa si nos piden obtener el resultado de la tabla derecha?

```
select
  political_party,
  case when gender = 'F' then 1 else 0 end as F,
  case when gender = 'M' then 1 else 0 end as M
from Tabla1;
```

political_party	gender
red	F
red	M
red	F
blue	F



political_party	F	M
red	1	0
red	0	1
red	1	0
blue	1	0

También lo podemos utilizar para sumar

```
select
  political_party,
  sum(case when gender = 'F' then 1 else 0 end) as F,
  sum(case when gender = 'M' then 1 else 0 end) as M
from Tabla1
group by political_party;
```

political_party	gender
red	F
red	M
red	F
blue	F



political_party	F	M
red	2	1
blue	1	0

Referencia: <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-case/>

- Common table expression (WITH)

Nos permite estructurar las consultas de una forma más legible. Cada consulta genera un resultado de todos temporal cómo si se tratara de un tabla pero está no persiste de forma física.

Es muy conveniente para complejas que deseamos estructurarlas de tal forma que fácil su interpretación. Es un principio de divide y vencerás.