Data Base Administrator PostgreSQL

Ernesto Quiñones A.
ernesto@eqsoft.net

Sesión
Manipulación de Datos (SQL)

PSQL

El comando básico para cualquier buen DBA en PostgreSQL es “psql”.

PSQL es un comando de consola que nos permitirá conectarnos a cualquier servidor PostgreSQL dentro del alcance de nuestra red, algunos parámetros a tomar en cuenta:

```
psql BASE_DE_DATOS [opciones]

-h, --host=HOST        <-- hostname ó ip del servidor al que se
                           conectará
-p, --port=PUERTO      <-- puerto de conexión (5432 por defecto)
-U, --username=USUARIO <-- nombre de usuario
-w, --no-password      <-- no solicita contraseña
-W, --password         <-- exige ingresar contraseña
```

PSQL

```
psql -U dbadmin -l                <-- Lista las bases de datos del
password for user dbadmin:            servidor sin ingresar a este
```

```
                                  List of databases
   Name    |  Owner   | Encoding |  Collation  |    Ctype    |Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres : postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres : postgres=CTc/postgres
```

Ejecutar un query de forma externa:

```
[user@host]# psql tipo2 -U admindb -c "select * from permisos";
Password for user admindb:
 idp | id
-----+----
(0 rows)
```

Ejecutar un query desde un file:

```
[ernesto@localhost ~]$ touch query.sql
[ernesto@localhost ~]$ echo "select * from permisos;" > query.sql
[ernesto@localhost ~]$ echo "select * from usuarios;" >> query.sql
[ernesto@localhost ~]$ cat query.sql
select * from permisos;
select * from usuarios;

[ernesto@localhost ~]$ psql tipo2 -U admindb -f query.sql
Contraseña para usuario admindb:
 idp | id       id | nombre
-----+----     ----+--------
(0 filas)      (0 filas)
```

PSQL

Grabar todo lo que se ejecuta durante la sesión:

```
[ernesto@localhost ~]$ touch log.txt
[ernesto@localhost ~]$ psql tipo2 -U admindb -L log.txt
Contraseña para usuario admindb:
psql (10.0)
Digite «help» para obtener ayuda.

tipo2=# \dt
        Listado de relaciones
 Esquema |  Nombre  | Tipo  |  Dueño
---------+----------+-------+---------
 public  | permisos | tabla | admindb
 public  | usuarios | tabla | admindb
(2 filas)

tipo2=# select * from permisos
tipo2-# ;
 idp | id
-----+----
(0 filas)

tipo2=# \q
[ernesto@localhost ~]$ cat log.txt
        Listado de relaciones
 Esquema |  Nombre  | Tipo  |  Dueño
---------+----------+-------+---------
 public  | permisos | tabla | admindb
 public  | usuarios | tabla | admindb
(2 filas)

********* QUERY **********
select * from permisos
;
**************************

 idp | id
-----+----
(0 filas)
```

PSQL

Una vez conectado a la base de datos podemos usar “\?” para pedir ayuda, por ejemplo con “\d” podemos pedir esta información:

```
\d[S+]            listar tablas, vistas y secuencias
\d[S+]  NOMBRE    describir tabla, índice, secuencia o vista
\da[S]  [PATRÓN]  listar funciones de agregación
\dA[+]  [PATRÓN]  listar métodos de acceso
\db[+]  [PATRÓN]  listar tablespaces
\dc[S+] [PATRÓN]  listar conversiones
\dC[+]  [PATRÓN]  listar conversiones de tipo (casts)
\dd[S]  [PATRÓN]  listar comentarios de objetos que no aparecen en otra parte
\dD[S+] [PATRÓN]  listar dominios
\ddp    [PATRÓN]  listar privilegios por omisión
\dE[S+] [PATRÓN]  listar tablas foráneas
\det[+] [PATRÓN]  listar tablas foráneas
\des[+] [PATRÓN]  listar servidores foráneos
\deu[+] [PATRÓN]  listar mapeos de usuario
\dew[+] [PATRÓN]  listar conectores de datos externos
\df[antw][S+] [PATRÓN]  listar funciones [sólo ag./normal/trigger/ventana]
\dF[+]  [PATRÓN]  listar configuraciones de búsqueda en texto
\dFd[+] [PATRÓN]  listar diccionarios de búsqueda en texto
\dFp[+] [PATRÓN]  listar analizadores (parsers) de búsq. en texto
\dFt[+] [PATRÓN]  listar plantillas de búsqueda en texto
\dg[S+] [PATRÓN]  listar roles
\di[S+] [PATRÓN]  listar índices
\dl               listar objetos grandes, lo mismo que \lo_list
\dL[S+] [PATRÓN]  listar lenguajes procedurales
\dm[S+] [PATRÓN]  listar vistas materializadas
\dn[S+] [PATRÓN]  listar esquemas
\do[S]  [PATRÓN]  listar operadores
\dO[S]  [PATRÓN]  listar ordenamientos (collations)
\dp     [PATRÓN]  listar privilegios de acceso a tablas, vistas y secuencias
\drds [PAT1 [PAT2]] listar parámetros de rol por base de datos
\dRp[+] [PATTERN]      list replication publications
\dRs[+] [PATTERN]      list replication subscriptions
\ds[S+] [PATRÓN]  listar secuencias
\dt[S+] [PATRÓN]  listar tablas
\dT[S+] [PATRÓN]  listar tipos de dato
\du[S+] [PATRÓN]  listar roles
\dv[S+] [PATRÓN]  listar vistas
\dx[+]  [PATRÓN]  listar extensiones
\dy     [PATRÓN]   listar disparadores por eventos
```

PSQL

Usar “S” muestra además los datos del sistema que se solicitan en la consulta, así por ejemplo:

```
pruebas=# \dt
        Listado de relaciones
Esquema  | Nombre | Tipo  |  Dueño
---------+--------+-------+----------
public   | tabla1 | tabla | postgres

pruebas=# \dtS
        Listado de relaciones
Esquema     |         Nombre          | Tipo  |  Dueño
------------+-------------------------+-------+----------
....... (presenta varias tablas del sistema)
pg_catalog  | pg_type                 | tabla | postgres
pg_catalog  | pg_user_mapping         | tabla | postgres
public      | tabla1                  | tabla | postgres
(43 filas)
```

PSQL

El sufijo “+” muestra información adicional a la presentada, por ejemplo descripciones (puede usarse en combinación con “S”)

```
pruebas=# \dt
        Listado de relaciones
Esquema  | Nombre | Tipo  |  Dueño
---------+--------+-------+----------
public   | tabla1 | tabla | postgres

pruebas=# \dt+
        Listado de relaciones
Esquema | Nombre | Tipo  |  Dueño   | Tamaño  | Descripción
---------+--------+-------+----------+---------+-------------
public  | tabla1 | tabla | postgres | 0 bytes |

                          Listado de relaciones
Esquema     |         Nombre          | Tipo  |  Dueño   |   Tamaño   | Descripción
------------+-------------------------+-------+----------+------------+-------------
....  (presenta varias tablas del sistema)
pg_catalog  | pg_ts_dict              | tabla | postgres | 8192 bytes |
pg_catalog  | pg_ts_parser            | tabla | postgres | 8192 bytes |
pg_catalog  | pg_ts_template          | tabla | postgres | 8192 bytes |
pg_catalog  | pg_type                 | tabla | postgres | 56 kB      |
pg_catalog  | pg_user_mapping         | tabla | postgres | 0 bytes    |
public      | tabla1                  | tabla | postgres | 0 bytes    |
(43 filas)
```

PSQL

\conninfo muestra información sobre la conexión a la base de datos de nuestra sesión, con \c cambiamos de base de datos, siempre y cuando nuestro usuario tenga derechos de acceso a la nueva base de datos que estamos especificando.

```
tipo2=# \conninfo
Está conectado a la base de datos «tipo2» como el usuario «admindb» a través del socket en
«/var/run/postgresql» port «5432».

tipo2=# \c template1
Ahora está conectado a la base de datos «template1» con el usuario «admindb».

template1=# \conninfo
Está conectado a la base de datos «template1» como el usuario «admindb» a través del socket en
«/var/run/postgresql» port «5432».
```

SQL

El SQL se divide en 2 tipos de comandos, los Data Manipulation Language (DML) y los Data Definition Languages (DDL).

Este último grupo son los clásicos Select, Insert, Update, Delete, requieren que el interprete de queries los analice y hasta reescriba para una optimización de su ejecución, esto se debe a que “mueven datos”.

El primer grupo corresponde a un conjunto de comandos que actúan directamente sobre datos, estructuras y objetos de la base de datos pero que no requieren de un análisis de ejecución, simplemente se ejecutan tal cual se escriben, por ejemplo CREATE, DROP, TRUNCATE, ALTER, etc.

SQL – Insert (nos permite ingresar datos a las tablas)

```
Insert into nombre_tabla (campos, ..) values ( valores, ..);
Insert into nombre_tabla values (valores, ..);
```

```
prueba2=# create table tablita(campo1 varchar, campo2 int);
CREATE TABLE

prueba2=# insert into tablita (campo1, campo2) values ('cadena1', 1);
INSERT 0 1

prueba2=# insert into tablita  values (2, 'cadena2');
ERROR:  la sintaxis de entrada no es válida para integer: «cadena2»
```

* Si no se especifica el orden en que debe insertarse los datos entonces se corre el riesgo de que los tipos de datos no concuerden y no se grabe los datos.

```
prueba2=# insert into tablita  values ('cadena3');
INSERT 0 1

prueba2=# insert into tablita values (1,'cadena');
ERROR:  invalid input syntax for integer: "cadena"
```

SQL - Insert

* No es necesario que se llenen todos los campos de una tabla, a menos que tengan la restricción NOT NULL

```
prueba2=# create table tablita2 (campo1 char(10), campo2 int not nu<ll);
CREATE TABLE

prueba2=# insert into tablita2 (campo1, campo2) values ('texto', 2);
INSERT 0 1

prueba2=# insert into tablita2 (campo1) values ('texto');
ERROR:  el valor null para la columna «campo2» viola la restricción
not null

prueba2=# insert into tablita2 (campo1, campo2) values ('texto', 0);
INSERT 0 1

prueba2=# insert into tablita2 (campo1, campo2) values ('texto', null);
ERROR:  el valor null para la columna «campo2» viola la restricción
not null

prueba2=# insert into tablita2 (campo1, campo2) values (null, 3);
INSERT 0 1
```

SQL - Insert

*** Es posible ingresar datos en grupo:**

```
prueba2=# insert into tablita2 (campo1, campo2) values
       ('A', 10), ('B', 20);  <-- puede añadir tantos datos como
                                  se desee
INSERT 0 2


prueba2=# insert into tablita2 (campo2) values
                                        (generate_series(1,500));
INSERT 0 500
prueba2=# select count(*) from tablita;
count
-------
 505
(1 row)
```

SQL - Insert

Cuando trabajamos con tablas con el tipo de datos “SERIAL” es posible retornar el valor autoincremental de este luego que se ejecuta un insert.

```
prueba=# create table tbl_simple ( id serial, nombre varchar(10));
   NOTICE:  CREATE TABLE creará una secuencia implícita
   «tbl_simple_id_seq» para la columna serial «tbl_simple.id»
   CREATE TABLE

prueba=# insert into tbl_simple (nombre) values ('ernesto')
           returning id;
     id
    ----
     1
prueba=# insert into tbl_simple (nombre) values ('juan') returning id;
     id
    ----
     2
```

SQL - Insert

Ejecutando un Insert por si solo, si sucede un error el SERIAL no aumenta su contador, pero esto puede verse afectado por transacciones, por lo tanto no se puede confiar que no se tendrá "huecos" entre los registros (se detallará al momento de ver transacciones).

```
prueba=# insert into tbl_simple (nombre) values
            ('estosupera10caracteres') returning id;
ERROR:  el valor es demasiado largo para el tipo character varying(10)


prueba=# insert into tbl_simple (nombre) values ('10chars') returning
id;
       id
      ----
       3
```

SQL - Insert

Es posible ejecutar una inserción de datos desde una consulta del tipo “Select”.

```
prueba=# create table tbl_simple2 ( id serial, nombre varchar(10));
    NOTICE:  CREATE TABLE creará una secuencia implícita
    «tbl_simple2_id_seq» para la columna serial «tbl_simple2.id»
    CREATE TABLE

prueba=# insert into tbl_simple2 ( nombre) select nombre from
         tbl_simple;
    INSERT 0 3

prueba=# select * from tbl_simple2;
   id | nombre
  ----+---------
    1 | ernesto
    2 | juan
    3 | 10chars
  (3 filas)
```

SQL – Llaves primarias

Una Llave primaria es uno o más campos que determinan un valor único en una tabla, no pueden haber 2 iguales o ser nulo (null), una tabla solo puede tener una llave primaria.

```
prueba2=# create table tablita3 (campo1 int primary key, campo2 float);
  NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito
  «tablita3_pkey» para la tabla «tablita3»
  CREATE TABLE

  prueba2=# \d tablita3;
         Tabla «public.tablita3»
   Columna |       Tipo       | Modificadores
  ---------+------------------+---------------
   campo1  | integer          | not null
   campo2  | double precision |
  Índices: "tablita3_pkey" PRIMARY KEY, btree (campo1)

  prueba2=# insert into tablita3 (campo1, campo2) values (1,2);
  INSERT 0 1
  prueba2=# insert into tablita3 (campo1, campo2) values (2,3);
  INSERT 0 1
  prueba2=# insert into tablita3 (campo1, campo2) values (2,4);
  ERROR:  llave duplicada viola restricción de unicidad «tablita3_pkey»
```

SQL – Llaves primarias

```
prueba2=# create table tablita4 (campo1 int, campo2 int, campo3 float,
            constraint pk_tablita4 primary key (campo1, campo2));
    NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito
    «pk_tablita4» para la tabla «tablita4»
    CREATE TABLE

prueba2=# \d tablita4;
          Tabla «public.tablita4»
 Columna |       Tipo       | Modificadores
---------+------------------+---------------
 campo1  | integer          | not null
 campo2  | integer          | not null
 campo3  | double precision |
Índices:
    "pk_tablita4" PRIMARY KEY, btree (campo1, campo2)

prueba2=# insert into tablita4 (campo1, campo2, campo3) values
(1,1,2);
INSERT 0 1
```

Una llave primaria puede estar compuesta por más de un campo de la tabla.

SQL – Llaves primarias

```
prueba2=# insert into tablita4 (campo1, campo2, campo3) values
(1,2,2);
   INSERT 0 1


prueba2=# insert into tablita4 (campo1, campo2, campo3) values
(1,1,3);
   ERROR:  llave duplicada viola restricción de unicidad «pk_tablita4»


prueba2=# insert into tablita4 (campo1, campo2, campo3) values
(1,2,4);
   ERROR:  llave duplicada viola restricción de unicidad «pk_tablita4»


prueba2=# insert into tablita4 (campo1, campo2, campo3) values
(1,null,4);
   ERROR:  el valor null para la columna «campo2» viola la restricción
           not null
```

SQL – Llaves Foraneas

Una llave foránea obliga a que el valor que se almacena en un campo de una tabla, este obligatoriamente registrado como dato de la llave primaria de una tabla "maestra", sino será rechazado, a esto se le llama integridad referencial.

```
prueba2=# create table maestro (campo1 int primary key, campo2 int);
    NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito
             «maestro_pkey» para la tabla «maestro»
    CREATE TABLE

prueba2=# create table detalle (campoA int primary key, campoB int,
constraint fk_detalle foreign key (campo) references maestro (campo2));
    NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito
             «detalle_pkey» para la tabla «detalle»
    ERROR:  no hay restricción unique que coincida con las columnas
            dadas en la tabla referida «maestro»

prueba2=# create table detalle (campoA int primary key, campoB int,
constraint fk_detalle foreign key (campoB) references maestro (campo1));
    NOTICE: CREATE TABLE / PRIMARY KEY creará el índice implícito
            «detalle_pkey» para la tabla «detalle»
    CREATE TABLE
```

SQL – Llaves Foraneas

```
prueba2=# \d detalle
      Tabla «public.detalle»
 Columna |  Tipo   | Modificadores
---------+---------+---------------
 campoa  | integer | not null
 campob  | integer |
Índices:
    "detalle_pkey" PRIMARY KEY, btree (campoa)
Restricciones de llave foránea:
    "fk_detalle" FOREIGN KEY (campob) REFERENCES maestro(campo1)
```

SQL – Llaves Foraneas

```
prueba2=# insert into detalle (campoA, campoB) values (1,2);
   ERROR:   inserción o actualización en la tabla «detalle» viola la
            llave foránea «fk_detalle»
   DETALLE: La llave (campob)=(2) no está presente en la tabla
            «maestro».

prueba2=# insert into maestro (campo1) values (2);
   INSERT 0 1

prueba2=# insert into detalle (campoA, campoB) values (1,2);
   INSERT 0 1

prueba2=# insert into detalle (campoA, campoB) values (1,1);

   ERROR:   inserción o actualización en la tabla «detalle» viola la
            llave foránea «fk_detalle»
   DETALLE: La llave (campob)=(1) no está presente en la tabla
            «maestro».

prueba2=# insert into detalle1_1 (campoA, campoB) values (2,null);
INSERT 0 1
```

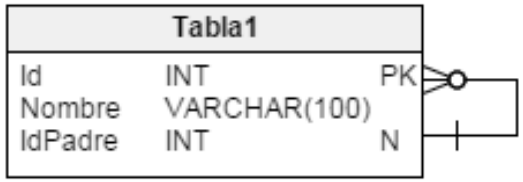
SQL – Llaves Foraneas

Eventualmente la referencia puede hacerse con un campo UNIQUE en la tabla maestra, un campo UNIQUE es como una llave primaria, no puede tener valores duplicados, pero una tabla puede tener varios campos UNIQUE.

```
prueba2=# create table maestro1 (campo1 int unique);
   NOTICE:  CREATE TABLE / UNIQUE creará el índice implícito
            «maestro1_campo1_key» para la tabla  «maestro1»
   CREATE TABLE
```

```
prueba2=# create table detalle1 (campoA int primary key, campoB int,
          constraint fk_detalle1 foreign key (campoB) references
          maestro1 (campo1));
   NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito
            «detalle1_pkey» para la tabla «detalle1»
   CREATE TABLE
```

SQL – Llaves Foraneas



Una tabla puede tener una referencia a si misma, esto es una referencia recursiva.

```
prueba2=# create table personas(id int,nombre varchar(100),padre int);
CREATE TABLE

prueba2=# alter table personas add primary key (id);
ALTER TABLE
                                                              ^
prueba2=# alter table personas add constraint fk_padre foreign key (padre)
references personas(id);
ALTER TABLE

prueba2=# insert into personas values(1,'abuelo',null);
prueba2=# insert into personas values(2,'padre',1);
prueba2=# insert into personas values(3,'hijo',2);

prueba2=# insert into personas values(4,'sobrino',21);
ERROR:  insert or update on table "personas" violates foreign key
constraint "fk_padre"
DETAIL:  Key (padre)=(21) is not present in table "personas".
```

SQL – Select

El comando SELECT permite ejecutar una consulta en una tabla y extraer datos de ella.

```
prueba2=# select * from tablita4;
 campo1 | campo2 | campo3
--------+--------+--------
      1 |      1 |      2
      1 |      2 |      2
(2 filas)

prueba2=# select  campo1, campo3 from tablita4;
 campo1 | campo3
--------+--------
      1 |      2
      1 |      2
(2 filas)
```

Se puede pedir todos los campos de una tabla con “*” o solo los que nos interesa detallando todos estosn entre “SELECT” y “FROM”.

SQL – Select - Joins

Es posible unir el resultado de 2 tablas utilizando los “joins”, en sus formas más básicas tenemos 2 opciones Join, Left Join y Right Join.

```
prueba2=# create table maestro ( codigo int, nombre varchar(100));
CREATE TABLE
prueba2=# create table detalle ( correlativo int, codigo int,
descripción varchar(100));
CREATE TABLE

prueba2=# insert into maestro values (1, 'plan a');
prueba2=# insert into maestro values (2, 'plan b');
prueba2=# insert into maestro values (3, 'plan c');

prueba2=# insert into detalle values (1, 1, 'detalle 1');
prueba2=# insert into detalle values (2, 1, 'detalle 2');
prueba2=# insert into detalle values (3, 2, 'detalle 3');
prueba2=# insert into detalle values (4, 4, 'detalle 4');
prueba2=# insert into detalle values (5, 4, 'detalle 5');
```

SQL – Select - Joins

```
prueba2=# select * from detalle, maestro;    <-- lista el
                                                 producto de ambas
                                                 Tablas

 correlativo | codigo | descripción | codigo | nombre
-------------+--------+-------------+--------+--------
           1 |      1 | detalle 1   |      1 | plan a
           2 |      1 | detalle 2   |      1 | plan a
           3 |      2 | detalle 3   |      1 | plan a
           4 |      4 | detalle 4   |      1 | plan a
           5 |      4 | detalle 5   |      1 | plan a
           1 |      1 | detalle 1   |      2 | plan b
           2 |      1 | detalle 2   |      2 | plan b
           3 |      2 | detalle 3   |      2 | plan b
           4 |      4 | detalle 4   |      2 | plan b
           5 |      4 | detalle 5   |      2 | plan b
           1 |      1 | detalle 1   |      3 | plan c
           2 |      1 | detalle 2   |      3 | plan c
           3 |      2 | detalle 3   |      3 | plan c
           4 |      4 | detalle 4   |      3 | plan c
           5 |      4 | detalle 5   |      3 | plan c
```

SQL – Select - Joins

```
select * from detalle
    join maestro on maestro.codigo = detalle.codigo; <-- lista la
                                                         Intersección
                                                         de ambas
                                                         Tablas
```

```
 correlativo | codigo | descripción | codigo | nombre
-------------+--------+-------------+--------+--------
           2 |      1 | detalle 2   |      1 | plan a
           1 |      1 | detalle 1   |      1 | plan a
           3 |      2 | detalle 3   |      2 | plan b
(3 filas)
```

SQL – Select - Joins

```
prueba2=# select * from detalle                    <-- lista todo lo
      left join maestro                                de “detalle” aunque
        on maestro.codigo = detalle.codigo;            no se intersecte con
                                                       “maestro”

 correlativo | codigo | descripción | codigo | nombre
-------------+--------+-------------+--------+--------
           1 |      1 | detalle 1   |      1 | plan a
           2 |      1 | detalle 2   |      1 | plan a
           3 |      2 | detalle 3   |      2 | plan b
           4 |      4 | detalle 4   |        |
           5 |      4 | detalle 5   |        |
(5 filas)
```

SQL – Select - Joins

```
prueba2=# select * from detalle              <-- lista todo lo de
          right join maestro                     "maestro" aunque
          on maestro.codigo = detalle.codigo;  no se  intersecte
                                                 con "detalle"

 correlativo | codigo | descripción | codigo | nombre
-------------+--------+-------------+--------+--------
           2 |      1 | detalle 2   |      1 | plan a
           1 |      1 | detalle 1   |      1 | plan a
           3 |      2 | detalle 3   |      2 | plan b
             |        |             |      3 | plan c
(4 filas)
```

Se pueden hacer "joins" entre 2 ó más tablas, pueden mezclarse los diversos tipos de joins presentados cuando se involucran consultas de muchas tablas.

Existe una variación llamada NATURAL JOIN, para que se de un NATURAL JOIN es necesario que dos tablas compartan un campo que las relacione pero que tenga el mismo nombre.

```
prueba2=# create table factura
              (numero int, monto numeric(10,2));
prueba2=# create table factura_detalle
              ( id int, numero int, producto varchar(100),
                valor numeric(10,2));

prueba2=# insert into factura values (1,100);
prueba2=# insert into factura values (2,200);

prueba2=# insert into factura_detalle values
              (1,1,'papas',50), (2,1,'camote',50);
prueba2=# insert into factura_detalle values
              (3,2,'pollo',100), (4,2,'carne',100);
```

```
prueba2=# select * from factura natural join factura_detalle;
 numero | monto  | id | producto | valor
--------+--------+----+----------+--------
      1 | 100.00 |  1 | papas    |  50.00
      1 | 100.00 |  2 | camote   |  50.00
      2 | 200.00 |  3 | pollo    | 100.00
      2 | 200.00 |  4 | carne    | 100.00

prueba2=# alter table factura rename column numero to numerdoid;

prueba2=# select * from factura natural join factura_detalle;
 numerdoid | monto  | id | numero | producto | valor
-----------+--------+----+--------+----------+--------
         1 | 100.00 |  1 |      1 | papas    |  50.00
         1 | 100.00 |  2 |      1 | camote   |  50.00
         1 | 100.00 |  3 |      2 | pollo    | 100.00
         1 | 100.00 |  4 |      2 | carne    | 100.00
         2 | 200.00 |  1 |      1 | papas    |  50.00
         2 | 200.00 |  2 |      1 | camote   |  50.00
         2 | 200.00 |  3 |      2 | pollo    | 100.00
         2 | 200.00 |  4 |      2 | carne    | 100.00
(8 rows)
```

FULL JOIN, permite cruzar la información de las tablas, especificando los registros que se intersectan y los que no de cada tabla involucrada en la consulta.

```
tipo2=# create table padre (idp int, nombrep varchar(100));
tipo2=# create table hijo (idh int, nombreh varchar(100), idp int);
tipo2=# insert into padre (idp, nombrep) values (1,'pedro'), (2, 'juan'), (3, 'alberto');
tipo2=# insert into hijo (idh, nombreh, idp) values (100, 'luigi',0), (200,'marco',1), (300,
'felipe',2), (400, 'ernesto',5);

tipo2=# select a.idh, a.nombreh, a.idp, b.nombrep from hijo a join padre b on a.idp = b.idp;
 idh | nombreh | idp | nombrep
-----+---------+-----+---------
 200 | marco   |   1 | pedro
 300 | felipe  |   2 | juan

tipo2=# select a.idh, a.nombreh, a.idp, b.nombrep from hijo a left join padre b on a.idp = b.idp;
 idh | nombreh | idp | nombrep
-----+---------+-----+---------
 100 | luigi   |   0 |
 200 | marco   |   1 | pedro
 300 | felipe  |   2 | juan
 400 | ernesto |   5 |

tipo2=# select a.idh, a.nombreh, a.idp, b.nombrep from hijo a right join padre b on a.idp = b.idp;
 idh | nombreh | idp | nombrep
-----+---------+-----+---------
 200 | marco   |   1 | pedro
 300 | felipe  |   2 | juan
     |         |     | alberto

tipo2=# select a.idh, a.nombreh, a.idp, b.nombrep from hijo a full join padre b on a.idp = b.idp;
 idh | nombreh | idp | nombrep
-----+---------+-----+---------
 100 | luigi   |   0 |
 200 | marco   |   1 | pedro
 300 | felipe  |   2 | juan
 400 | ernesto |   5 |
     |         |     | alberto
```

A un JOIN podemos aplicarle filtros para extraer los datos que nos interesan.

```
prueba2=# select * from factura
             join factura_detalle
               on factura.numeroid = factura_detalle.numero
                  and factura.numero =1;

numero | monto  | id | numero | producto | valor
-------+--------+----+--------+----------+-------
     1 | 100.00 |  1 |      1 | papas    | 50.00
     1 | 100.00 |  2 |      1 | camote   | 50.00
```

SQL – Select – Alias

Es posible en una consulta colocar alias a los campos y a las tablas para mejorar los resultados de las consultas y eventualmente las uniones de tablas.

```
prueba2=# select codigo as Cmaestro, nombre Nmaestro from maestro;
 cmaestro | nmaestro
----------+----------
        1 | plan a
        2 | plan b
        3 | plan c
(3 filas)

prueba2=# select maestro.codigo as Cmaestro, nombre Nmaestro from
maestro
join detalle b on b.codigo = maestro.codigo;
 cmaestro | nmaestro
----------+----------
        1 | plan a
        1 | plan a
        2 | plan b
(3 filas)
```

SQL – Select – Alias

```
prueba2=# select maestro.codigo as Cmaestro, nombre Nmaestro,
b.codigo as Dmaestro from maestro
join detalle b on b.codigo = maestro.codigo;
 cmaestro | nmaestro | dmaestro
----------+----------+----------
        1 | plan a   |        1
        1 | plan a   |        1
        2 | plan b   |        2
(3 filas)
```

Cuando tenemos dos tablas con campos con el mismo nombre y no especificamos la talba a la que pertenece al incluirlo en un select la base de datos nos dará un error porque no sabe de donde tomarla.

```
prueba2=# select maestro.codigo, nombre, codigo from maestro
              join detalle b on b.codigo = maestro.codigo;

ERROR:  la referencia a la columna «codigo» es ambigua
```

Select – Distinct

Permite generar un resultado de datos único al realizar una consulta.

```
prueba2=# create table tablaA (campo1 int, campo2 varchar(10));
prueba2=# insert into tablaA values (1, 'A');
prueba2=# insert into tablaA values (2, 'A');
prueba2=# insert into tablaA values (2, 'B');
prueba2=# insert into tablaA values (2, 'B');
prueba2=# insert into tablaA values (1, 'C');

prueba2=# select * from tablaA;
 campo1 | campo2
--------+--------
      1 | A
      2 | A
      2 | B
      2 | B
      1 | C
(5 filas)
```

Select – Distinct

```
prueba2=# select distinct * from tablaa;
 campo1 | campo2
--------+--------
      2 | A
      1 | C
      1 | A
      2 | B
(4 filas)

prueba2=# select distinct campo1 from tablaa;
 campo1
--------
      1
      2
(2 filas)
```

SQL – Select – Where

La clausula WHERE nos permitirá filtrar los datos que debe devolver la consulta.

```
prueba2=# select * from tablaA where campo2 = 'A';
 campo1 | campo2
--------+--------
      1 | A
      2 | A
(2 filas)

prueba2=# select * from tablaA where campo2 = '';
 campo1 | campo2
--------+--------
(0 filas)
```

SQL – Select – Where

```
prueba2=# insert into tablaA values (1, null);
INSERT 0 1

prueba2=# select * from tablaA where campo2 = null; ←
 campo1 | campo2
--------+--------
(0 filas)
```

los campos con tipo de datos “null” requieren una sintaxis especial

```
prueba2=# select * from tablaA where campo2 is null;
 campo1 | campo2
--------+--------
      1 |
(1 fila)
```

SQL – Select – Where

WHERE nos permite simular JOINs.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
(3 filas)

prueba2=# select * from detalle;
 correlativo | codigo | descripción
-------------+--------+-------------
           1 |      1 | detalle 1
           2 |      1 | detalle 2
           3 |      2 | detalle 3
           4 |      4 | detalle 4
           5 |      4 | detalle 5
(5 filas)
```

SQL – Select – Where

WHERE nos permite simular JOINs.

```
prueba2=# select * from maestro a , detalle b
                where a.codigo = b.codigo;

 codigo | nombre | correlativo | codigo | descripción
--------+--------+-------------+--------+-------------
      1 | plan a |           2 |      1 | detalle 2
      1 | plan a |           1 |      1 | detalle 1
      2 | plan b |           3 |      2 | detalle 3
(3 filas)
```

SQL – Select – Where

WHERE nos permite manejar operadores lógicos, los más usados son AND, OR y operadores de precisión como =, !=, <, <=, >, >= *(hay muchos otros operadores lógicos y de precisión que se detallarán más adelante)*

```
prueba2=# select * from maestro a , detalle b
             where a.codigo = b.codigo and correlativo < 3;
 codigo | nombre | correlativo | codigo | descripción
--------+--------+-------------+--------+-------------
      1 | plan a |           2 |      1 | detalle 2
      1 | plan a |           1 |      1 | detalle 1
(2 filas)

prueba2=# select * from detalle
             where codigo < 3 or  descripción = 'detalle 5';
 correlativo | codigo | descripción
-------------+--------+-------------
           1 |      1 | detalle 1
           2 |      1 | detalle 2
           3 |      2 | detalle 3
           5 |      4 | detalle 5
(4 filas)
```

SQL – Select – Where

Podemos agrupar los operadores entre ( )

```
prueba2=# select * from detalle  where codigo = 2 and codigo = 1 or
descripción = 'detalle 5';
 correlativo | codigo | descripción
-------------+--------+-------------
           5 |      4 | detalle 5
(1 fila)

prueba2=# select * from detalle  where codigo = 2 and (codigo = 1 or
descripción = 'detalle 5');
 correlativo | codigo | descripción
-------------+--------+-------------
(0 filas)
```

SQL – Select – Where – Like e iLike

Like e iLike nos permite ejecutar búsquedas dentro de cadenas de caracteres.

```
prueba2=> select * from maestro where nombre like 'pl%';
codigo | nombre                          Todo lo que empiece con “pl”
--------+--------
     1 | plan a
     2 | plan b
     3 | plan c
(3 rows)

prueba2=> select * from maestro where nombre like '%b';
codigo | nombre                          Todo lo que termine con “b”
--------+--------
     2 | plan b
(1 row)

prueba2=> select * from maestro where nombre like '%c%';
codigo | nombre                          Todo lo que incluya “c” en
--------+--------                       cualquier posición
     3 | plan c
(1 row)
```

SQL – Select – Where – Like e iLike

Like e iLike se diferencia en que Like hace diferencias entre mayúsculas y minúsculas, mientras que iLike no, pero el tiempo de respuesta es menor.

```
prueba3=> select * from maestro where nombre like '%C%';
codigo | nombre
--------+--------
(0 rows)
```

```
prueba3=> select * from maestro where nombre ilike '%C%';
codigo | nombre
--------+--------
     3 | plan c
(1 row)
```

SQL – Select – Where – BETWEEN

BETWEEN nos permite hacer búsquedas de rangos, tanto de números como letras y fechas.

```
prueba3=> SELECT * FROM Productos WHERE
                                precio BETWEEN 10 AND 20;

prueba3=> SELECT * FROM Productos WHERE
                                precio NOT BETWEEN 10 AND 20;

prueba3=> SELECT * FROM Productos WHERE
                                fecha_ingreso
                                BETWEEN '2010-12-01' AND '2010-12-31';

prueba3=> SELECT * FROM Productos WHERE
                                substr(NOMBRE,1,1) BETWEEN 'A' and 'G';
```

SQL – Select – Union

UNION nos permite unir los resultados de 2 consultas, como restricción los resultados deben coincidir en tipo de datos para las columnas especificadas en cada consulta, se mostrarán solo los datos únicos, si algún registro se repite de se presentará solo una vez.

```
prueba2=# select * from tablaA;
 campo1 | campo2
--------+--------
      1 | A
      2 | A
      2 | B
      2 | B
      1 | C
      1 |
                                      ^
prueba2=# select campo1, campo2 from tablaA where campo1 = 1
          union
          select campo1, campo2 from tablaA where campo2 = 'B';
 campo1 | campo2
--------+--------
      1 | A
      2 | B
      1 | C
      1 |
```

SQL – Select – Union

UNION ALL une las tablas y visualiza todos los registros de ellas, aunque esten repetidos, estos se mostrarán tantas veces como hayan sido registrados.

```
prueba2=# select campo1, campo2 from tablaA where campo1 = 1
          union all
          select campo1, campo2 from tablaA where campo2 = 'B';

 campo1 | campo2
--------+--------
      1 | A
      1 | C
      1 |
      2 | B
      2 | B
```

SQL – Select – Intersec y Except

Como UNION unen 2 o más consultas, pero en este caso con INTERSEC se mostrarán en el resultado solo aquellos datos que coincidan entre ambas consultas y en EXCEPT los que no coincidan.

```
pruebas=# select * from tablaA;
 campo1 | campo2
--------+--------
      1 | A
      2 | A
      2 | B
      2 | B
      1 | C
      1 |

pruebas=# select campo1, campo2 from tablaA
          intersect
          select campo1, campo2 from tablaA where campo1=2;
 campo1 | campo2
--------+--------
      2 | B
      2 | A
```

SQL – Select – Intersec y Except

```
pruebas=# select campo1, campo2 from tablaA
          except
          select campo1, campo2 from tablaA where campo1=2;

 campo1 | campo2
--------+--------
      1 | C
      1 | A
      1 |
```

SQL – Select – GROUP BY

GROUP nos permite agrupar los resultados de una consulta, generalmente se usa en operaciones matemáticas como suma, promedio, mayor valor, menor valor, etc.

```
prueba2=# select * from tablaA;
 campo1 | campo2
--------+--------
      1 | A
      2 | A
      2 | B
      2 | B
      1 | C
      1 |

prueba2=# select sum(campo1), avg(campo1), campo2
             from tablaA group by campo2;
 sum |          avg           | campo2
-----+------------------------+--------
   1 | 1.0000000000000000000 |
   4 |     2.000000000000000 | B
   1 | 1.0000000000000000000 | C
   3 |     1.500000000000000 | A
```

SQL – Select – GROUP BY

```
prueba2=# select min(campo1), max(campo1), campo2
             from tablaA group by campo2;
 min | max | campo2
-----+-----+--------
   1 |   1 |
   2 |   2 | B
   1 |   1 | C
   1 |   2 | A
```

* GROUP siempre requerirá que el campo especificado en “group by” sea parte de los resultados de la consulta.

SQL – Select – GROUP BY

Los groups utilizan generalmente funciones que realizan algún tipo de cálculo y devuelven un resultado único, este resultado se computará por las agrupaciones creadas en el GROUP BY.

Estas funciones se llaman funcioens aagregadas y pueden ser consultadas aquí:
http://www.postgresql.org/docs/9.4/static/functions-aggregate.html

Las funciones más comunes son:

- **AVG ← Promedio**
- **Count ← Contrar filas procesadas**
- **SUM ← Sumar valores**
- **MIN ← Valor mínimo**
- **MAX ← Valor máximo**

SQL – Select – GROUP BY y HAVING

HAVING dentro de un GROUP nos permite filtrar los resultados de la consulta operando funciones en el filtro.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
```

```
prueba2=# select codigo, sum(codigo) , count(*) from maestro
            where sum(codigo) > 1 group by codigo;

ERROR:  no se permiten funciones de agregación en la cláusula WHERE
LÍNEA 1: ...odigo, sum(codigo) , count(*) from maestro  where
sum(codigo...
```

SQL – Select – GROUP BY y HAVING

```
prueba2=# select codigo, sum(codigo) , count(*)
            from maestro group by codigo
               having sum(codigo) > 1;

 codigo | sum | count
--------+-----+-------
      3 |   6 |     2
      2 |   2 |     1
(2 filas)

prueba2=# select codigo, sum(codigo) , count(*), nombre
             from maestro group by codigo, nombre
                having sum(codigo) > 1;

 codigo | sum | count | nombre
--------+-----+-------+--------
      3 |   6 |     2 | plan c
      2 |   2 |     1 | plan b
(2 filas)
```

SQL – Select – ORDER BY, LIMIT y OFFSET

ORDER BY nos permite ordenar el resultado de nuestra consulta de manera ASCendente y DESCendente, en vez de indicar el nombre del campo de ordenamiento también podríamos simplemete referencias el número de su posición.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
      1 | plan a
prueba2=# select * from maestro order by nombre asc;
 codigo | nombre
--------+--------
      1 | plan a
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
```

SQL – Select – ORDER BY, LIMIT y OFFSET

```
prueba2=# select codigo, nombre from maestro order by 2 desc;
 codigo | nombre
--------+--------
      3 | plan c
      3 | plan c
      2 | plan b
      1 | plan a
      1 | plan a
```

* OjO PostgreSQL presenta los datos conforme los obtiene al armar el resultado de la consulta, sin un ordenamiento entonces el resultado podría ser aleatorio.

SQL – Select – ORDER BY, LIMIT y OFFSET

LIMIT permite restringir la cantidad de registros a presentar en una consulta y OFFSET permite indicar a partir de que posición de resultado presentar el resultado, son efectivos solo si se usan con ORDER BY.

```
prueba2=# select * from maestro order by codigo;
 codigo | nombre
--------+--------
      1 | plan a
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
(5 rows)

prueba2=# select * from maestro order by codigo limit 2;
 codigo | nombre
--------+--------
      1 | plan a
      1 | plan a
(2 rows)
```

SQL – Select – ORDER BY, LIMIT y OFFSET

```
prueba2=# select * from maestro order by codigo limit 2 offset 2;
 codigo | nombre
--------+--------
      2 | plan b
      3 | plan c
(2 rows)
```

SQL – Select – SUBQUERYS

Una manera de facilitar la lógica de las consultas es crear sub-consultas dentro de una consulta principal, a esto se le conoce como subquerys, la combinación del uso de está funcionalidad puede ser muy variada dado que una sub-consulta puede tener a su vez otras sub-consultas anidadas.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c

prueba2=# select * from detalle;
 correlativo | codigo | descripción
-------------+--------+-------------
           1 |      1 | detalle 1
           2 |      1 | detalle 2
           3 |      2 | detalle 3
           4 |      4 | detalle 4
           5 |      4 | detalle 5
```

SQL – Select – SUBQUERYS

Siempre debe de tomar en cuenta que los SUBQUERYS añaden mucho consumo computacional a sus consultas ya que se deben procesar las consultas por cada registro afectado por que QUERY principal.

```
prueba2=# select * from detalle a where
            a.codigo in (select b.codigo from maestro b);

 correlativo | codigo | descripción
-------------+--------+-------------
           1 |      1 | detalle 1
           2 |      1 | detalle 2
           3 |      2 | detalle 3

prueba2=# select * from detalle a where
            a.codigo in
               (select b.codigo from maestro b where b.codigo > 1);

 correlativo | codigo | descripción
-------------+--------+-------------
           3 |      2 | detalle 3
```

SQL – Select – SUBQUERYS

```
prueba2=# select a.correlativo, a.descripción, a.codigo,
   (select b.nombre from maestro b
       where b.codigo=a.codigo) as nombre_maestro
   from detalle a;
 correlativo | descripción | codigo | nombre_maestro
-------------+-------------+--------+----------------
           1 | detalle 1   |      1 | plan a
           2 | detalle 2   |      1 | plan a
           3 | detalle 3   |      2 | plan b
           4 | detalle 4   |      4 |
           5 | detalle 5   |      4 |

prueba2=# select a.correlativo, a.descripción,
   ( select sum(b.codigo) from maestro b ) total_codigo,
   ( select c.codigo from maestro c
        where a.codigo = c.codigo) as codigo
   from detalle a;
 correlativo | descripción | total_codigo | codigo
-------------+-------------+--------------+--------
           1 | detalle 1   |            6 |      1
           2 | detalle 2   |            6 |      1
           3 | detalle 3   |            6 |      2
           4 | detalle 4   |            6 |
           5 | detalle 5   |            6 |
```

SQL – Select – SUBQUERYS

```
select correlativo, descripción, total_codigo, codigo,
       total_codigo/codigo as division
   From (
        select a.correlativo, a.descripción,
           (select sum(b.codigo) from maestro b) total_codigo,
           ( select c.codigo from maestro c
                where a.codigo = c.codigo) as codigo
        from detalle a)
   as tabla_temporal;
```

correlativo	descripción	total_codigo	codigo	division
1	detalle 1	6	1	6
2	detalle 2	6	1	6
3	detalle 3	6	2	3
4	detalle 4	6		
5	detalle 5	6		

SQL – Update

Permite actualizar los datos de una tabla, se puede actualizar la información haciendo uso de el filtro WHERE para solo afectar algunos datos.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
      1 | plan a

prueba2=# update maestro set nombre = 'plan a1' where codigo = 1;

prueba2=# select * from maestro;
 codigo | nombre
--------+---------
      2 | plan b
      3 | plan c
      3 | plan c
      1 | plan a1
      1 | plan a1
```

SQL – Update

```
pruebas=# update maestro set nombre = 'A-' || nombre;
UPDATE 5
pruebas=# select * from maestro;
 codigo |  nombre
--------+----------
      1 | A-plan a
      2 | A-plan b
      3 | A-plan c
      3 | A-plan c
      1 | A-plan a

pruebas=# update maestro set nombre = 'B-' || (select nombre from
maestro a where maestro.codigo = a.codigo limit 1);
UPDATE 5
pruebas=# select * from maestro;
 codigo |   nombre
--------+------------
      1 | B-A-plan a
      2 | B-A-plan b
      3 | B-A-plan c
      3 | B-A-plan c
      1 | B-A-plan a
```

SQL – Update

```
pruebas=# select * from maestro2;
 codigo | nombre
--------+--------
      1 | C1
      2 | C2
      3 | C3
      3 | C3
      1 | C1

pruebas=# update maestro set nombre = a.nombre from
            (select codigo, nombre from maestro2) as a
          where maestro.codigo = a.codigo;

pruebas=# select * from maestro2;
 codigo | nombre
--------+--------
      1 | C1
      2 | C2
      3 | C3
      3 | C3
      1 | C1
```

SQL – Update

Un UPDATE puede retornar un set de datos al estilo de una consulta usando el comando RETURNING

```
pruebas=# update maestro set nombre = 'D-' || a.nombre
          from (select codigo, nombre from maestro2) as a
          where maestro.codigo = a.codigo returning maestro.*;

 codigo | nombre
--------+--------
      1 | D-C1
      1 | D-C1
      2 | D-C2
      3 | D-C3
      3 | D-C3
```

SQL – Delete

Permite eliminar los datos de una tabla, se puede eliminar la información haciendo uso de el filtro WHERE para solo afectar algunos datos.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
      1 | plan a

prueba2=# delete from maestro ' where codigo = 1;
DELETE 2

prueba2=# select * from maestro;
 codigo | nombre
--------+---------
      2 | plan b
      3 | plan c
      3 | plan c
```

SQL – Delete

Podemos llevar a cabo un DELETE haciendo un WHERE que involucre a otra tabla usando USING, el uso de RETURNING también es válido en DELETE como en el UPDATE.

```
pruebas=# select * from maestro;
 codigo | nombre
--------+--------
      1 | D-C1
      1 | D-C1
      2 | D-C2
      3 | D-C3
      3 | D-C3

pruebas=# delete from maestro using maestro2 where
            maestro.codigo = maestro2.codigo and maestro2.codigo > 2;

pruebas=# delete from maestro using maestro2 where
            maestro.codigo = maestro2.codigo returning maestro.*;
 codigo | nombre
--------+--------
      1 | D-C1
      1 | D-C1
      2 | D-C2
```

SQL – Delete

```
pruebas=# select * from maestro;
 codigo | nombre
--------+--------
(0 filas)
```

SQL – Truncate

Permite eliminar todos los datos de una tabla, se diferencia de DELETE en que esta sentencia es “command utility”, se ejecuta directamente y es mucho más conveniente cuando el borrado debe ser masivo, demora mucho menos que un DELETE sin filtro WHERE.

```
prueba2=# select * from maestro;
 codigo | nombre
--------+--------
      1 | plan a
      2 | plan b
      3 | plan c
      3 | plan c
      1 | plan a
(5 rows)

prueba2=# truncate maestro;
TRUNCATE TABLE

prueba2=# select * from maestro;
(0 rows)
```