Programación en SQL con PostgreSQL

Francisco Alonso Sarría

1 Introducción

El lenguaje estructurado de consultas (SQL) es un lenguaje de base de datos normalizado, utilizado por la gran mayoría de los servidores de bases de datos que manejan bases de datos relacionales u objeto-relacionales.

Es un **lenguaje declarativo** en el que las órdenes especifican cual debe ser el resultado y no la manera de conseguirlo (como ocurre en los **lenguajes procedimentales**). Al ser declarativo es muy sistemático, sencillo y con una curva de aprendizaje muy agradable ya que sus palabras clave permiten escribir las ordenes como si fueran frases en las que se especifica (en inglés) que es lo que queremos obtener. Por ejemplo:

SELECT nombre FROM municipios WHERE poblacion>5000 ORDER BY poblacion;

Devuelve el nombre de aquellos municipios con una población mayor de 5000 habitantes y los presenta ordenados por tamaño. Sin embargo los lenguajes declarativos carecen de la potencia de los procedimentales

Se ha convertido, debido a su eficiencia, en un estandar para las bases de datos relacionales, de hecho el gran éxito del modelo de base de datos relacional se debe en parte a la utilización de un lenguaje como SQL. A pesar de su tesórico caracter estandar, se han desarrollado, sobre una base común, diversas versiones ampliadas como las de Oracle o la de Microsoft SQL server. Incluye diversos tipos de capacidades:

- Comandos para la **definición y creación** de una base de datos (create table).
- Comandos para **inserción, borrado o modificación** de datos (insert, delete, update).
- Comandos para la **consulta** de datos seleccionados de acuerdo a criterios complejos que involucran diversas tablas relacionadas por un campo común (select).
- Capacidades aritméticas: En SQL es posible incluir operaciones aritméticas así como comparaciones, por ejemplo $A > B + 3$.
- **Asignación y comandos de impresión**: es posible imprimir una tabla construida por una consulta o almacenarla como una nueva tabla.

- **Funciones de agregación**: Operaciones tales como promedio (average), suma (sum), máximo (max), etc. se pueden aplicar a las columnas de una tabla para obtener una cantidad única y, a su vez, incluirla en consultas más complejas.

En una base de datos relacional, los resultados de la consulta van a ser datos individuales, tuplas[1] o tablas generados a partir de consultas en las que se establecen una serie de condiciones basadas en valores numéricos. Por ejemplo una típica consulta sobre una tabla en una base de datos relacional, utilizando SQL podría ser:

bd=# SELECT id, nombre, pob1991
FROM municipios
WHERE pob1991>20000;[2]

el resultado será una tabla en la que tendremos tres columnas (id, nombre, poblacion) procedentes de la tabla municipios, las filas corresponderán sólo a aquellos casos en los que la poblacion en 1991 (columna pob1991) sea mayor que 20000. En el caso de que sólo uno de los municipios cumpliera la condición obtendríamos una sola fila y en caso de que la consulta fuera:

bd=# SELECT pob1991
FROM municipios
WHERE pob1991>20000;

obtendríamos un sólo número, la población del municipio más poblado.

1.1 Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

1.2 Comandos

Existen dos tipos de comandos SQL:

- Los que permiten crear y definir nuevas bases de datos, campos e índices.

  CREATE Utilizado para crear nuevas tablas, campos e índices

  DROP Empleado para eliminar tablas e índices

[1] equivalente a una fila de una tabla

[2] A partir de este momento, cuando escriba una sentencia SQL lo haré en negrita, utilizando el prompt de PostgreSQL que consiste en el nombre de la base de datos (si no se especifica ninguna utilizare de forma genérica **bd**) seguido de **=#**, y con los diferentes elementos de la consulta separados por lineas. Esto último facilita la interpretación de la orden, pero recuerda que a la hora de trabajar es preferible escribir toda la orden en una sola linea

ALTER Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

- Los que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

  SELECT Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
  INSERT Utilizado para cargar lotes de datos en la base de datos en una única operación.
  UPDATE Utilizado para modificar los valores de los campos y registros especificados
  DELETE Utilizado para eliminar registros de una tabla de una base de datos

1.3 Cláusulas

Las cláusulas son condiciones utilizadas para concretar que datos son los que se desea seleccionar o manipular.

FROM Utilizada para especificar la tabla de la cual se van a seleccionar los registros

WHERE Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar

GROUP BY Utilizada para clasificar los registros seleccionados en grupos específicos

HAVING Utilizada para expresar la condición que debe satisfacer cada grupo

ORDER BY Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

1.4 Operadores Lógicos

AND Evalua dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

OR Evalúa dos condiciones y devuelve un valor de verdar si alguna de las dos es cierta.

NOT Devuelve el valor contrario de la expresión.

1.5 Operadores de Comparación

< Menor que

> Mayor que

<> Distinto de

<= Menor ó Igual que

>= Mayor ó Igual que

= Igual que

BETWEEN Utilizado para especificar un intervalo de valores.

o LIKE Para la comparación de una cadena de texto con una *expresión regular*

1.6 Funciones de Agregación

Las funciones de agregación se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

AVG Utilizada para calcular el promedio de los valores de un campo determinado

COUNT Utilizada para devolver el número de registros de la selección

SUM Utilizada para devolver la suma de todos los valores de un campo determinado

MAX Utilizada para devolver el valor más alto de un campo especificado

MIN Utilizada para devolver el valor más bajo de un campo especificado

2 Bases de datos relacionales

Es el modelo más utilizado hoy en día. Una base de datos relacional es básicamente un conjunto de tablas, similares a las tablas de una hoja de cálculo, formadas por filas (registros) y columnas (campos). Los registros representan cada uno de los objetos descritos en la tabla y los campos los atributos (variables de cualquier tipo) de los objetos. En el modelo relacional de base de datos, las tablas comparten algún campo entre ellas. Estos campos compartidos van a servir para establecer relaciones entre las tablas que permitan consultas complejas (figura 1). En esta figura aparecen tres tablas con información municipal, en la primera aparecen los nombres de los municipios, en la segunda el porcentaje en cada municipio de los diferentes usos del suelo y en la tercera la población en cada municipio lo largo del siglo XX. Como campo común aparece **ident**, se trata de un identificador numérico, único para cada municipio[3]

La idea básica de las bases de datos relacionales es la existencia de *entidades* (filas en una tabla) caracterizadas por *atributos* (columnas en la tabla). Cada tabla almacena entidades del mismo tipo y entre entidades de distinto tipo se establecen *relaciones*[4]. Las tablas comparten algún campo entre ellas, estos campos compartidos van a servir para establecer relaciones entre las tablas. Los atributos pueden ser de unos pocos tipos simples:

[3] Es preferible utilizar valores numéricos en lugar de una cadena de caracteres ya que se ahorra espacio y se evitan problemas con el uso de mayúsculas, acentos, etc.

[4] En la bibliografía inglesa sobre bases de datos se habla de *relations* (tablas) y *relationships* relaciones entre las tablas. El término base de datos relacional hace en realidad referencia a la organización de los datos en forma de tablas, no a las relaciones entre ellas

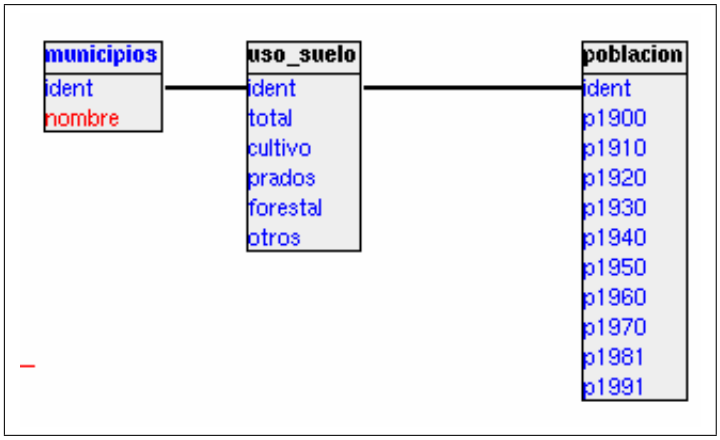

Figure 1: Esquema de base de datos relacional

- Números enteros
- Números reales
- Cadena de caracteres de longitud variable

Estos tipos simples se denominan *tipos atómicos* y permiten una mayor eficacia en el manejo de la base de datos pero a costa de reducir la flexibilidad a la hora de manejar los elementos complejos del mundo real y dificultar la gestión de datos espaciales, en general suponen un problema para cualquier tipo de datos geométricos.

Las relaciones que se establecen entre los diferentes elementos de dos tablas en una base de datos relacional pueden ser de tres tipos distintos:

- **Relaciones uno a uno**, se establecen entre una entidad de una tabla y otra entidad de otra tabla. Un ejemplo aparece en la figura 1.
- **Relaciones uno a varios**, se establecen entre varias entidades de una tabla y una entidad de otra tabla. Un ejemplo sería una tabla de pluviómetros en la que se indicara el municipio en el que se encuentra. La relación sería entre un municipio y varios pluviómetros
- **Relaciones varios a varios**, se establecen entre varias entidades de cada una de las tablas. Un ejemplo sería una tabla con retenes de bomberos y otra con espacios naturales a los que cada uno debe acudir en caso de incendio.

3 Entrada en el cliente y exploración de la base de datos

La gestión de bases de datos se basa en la existencia de un **programa servidor**; que organiza los datos, recibe las consultas, las ejecuta y las devuelve; y un **programa cliente** que el usuario ejecuta y que lanza las consultas creadas por este al servidor. El programa cliente y el servidor no tienen siquiera porque ejecutarse en el mismo ordenador.

Existen diferentes clientes para conectar al servidor de bases de datos de **PostgreSQL**. Vamos a utilizar en principio uno sencillo (**psql**). Si tecleamos:

psql -l

obtendremos un listado de todas la bases de datos disponibles para el servidor. Si queremos conectarnos a una de ellas se le especificará al teclear el comando:

psql clima

En este caso hemos especificado la base de datos a la que queremos conectarnos. El mensaje de bienvenida de psql será algo parecido a:

```
Welcome to psql 7.3.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

clima=#
```

Disponemos de una serie de comandos formados por una barra y una letra que realizan operaciones sencillas:

- \h para pedir ayuda sobre comandos SQL
- \? para pedir ayuda sobre los comandos de *barra y letra*
- \q para salir del programa
- \d para obtener un listado de las tablas que forman la base de datos

Como ves tenemos 3 tipos de variables. El tipo *int4* corresponde a números enteros, el tipo *float8* corresponde a numeros reales y *varchar* a cadenas de caracteres. De todos estos atributos el más importante es **ident** ya que asigna a cada municipio un identificador único que coincide con el identificador del polígono correspondiente a dicho municipio en el mapa vectorial.

- Tabla **observatorios**

```
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 indentinm | character varying(6)  |
 nombre    | character varying(50) |
 x         | integer               |
 y         | integer               |
 z         | smallint              |
 ident     | integer               |
 obs       | integer               |
```

- Tabla **menspluv**

```
 Column |         Type         | Modifiers
--------+----------------------+-----------
 ide    | character varying(6) |
 mes    | smallint             |
 ano    | smallint             |
 pluv   | real                 |
 ndias  | smallint             |
 max    | real                 |
 obs    | integer              |
```

- Tabla **menstem**

```
 Column  |         Type         | Modifiers
---------+----------------------+-----------
 ide     | character varying(6) |
 mes     | smallint             |
 ano     | smallint             |
 tmaxabs | real                 |
 tmaxmed | real                 |
 tmed    | real                 |
 tminmed | real                 |
 tminabs | real                 |
```

Como ves tenemos 4 tipos de variables. El tipo *integer* (4 bytes) corresponde a números enteros, el tipo *smallint* (2 bytes) corresponde a números enteros lo suficientemente pequeños como para necesitar sólo 2 bytes, el tipo *real* (8 bytes) corresponde a numeros reales y *character* a cadenas de caracteres especificándose en cada caso el número de caracteres (bytes) que ocupa. De todos estos atributos el más importante es **ide** (en la tabla observatorios se llama **indentinm** ya que asigna a cada observatorio un identificador único que coincide en todas las tablas.

4 Consultas de Selección

Las consultas de selección se utilizan para indicar al servidor de base de datos que devuelva información de las bases de datos, tal como se ha visto esta información devuelta puede ser un valor, una tupla o una tabla. A partir de este momento todos los ejemplos se refieren a la base de datos **clima**.

4.1 Consultas básicas

La sintaxis básica de una consulta de selección es la siguiente:

clima=# SELECT *campos*
FROM *tabla*;

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
clima=# SELECT nombre,x,y,z
FROM observatorios;
```

Esta consulta devuelve una tabla con el campo nombre y teléfono de la tabla clientes. La tabla devuelta no está almacenada en la base de datos, y por tanto no podrá ser objeto de posteriores consultas, salvo que la guardes de forma explícita con la orden **SELECT INTO**.

```
clima=# SELECT nombre,x,y,z
INTO resumen FROM observatorios;
```

de esta manera se genera una nueva tabla que contiene sólo las cuatro columnas seleccionadas.

4.2 Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la claúsula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
clima=# SELECT nombre,x,y,z
FROM observatorios
ORDER BY z;
```

Esta consulta devuelve los nombres de los observatorios junto a sus coordenadas pero ahora ordenados en función de su altitud.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
clima=# SELECT nombre,x,y,z
FROM observatorios
ORDER BY x,y;
```

Incluso se puede especificar el orden de los registros: ascendente mediante la claúsula (ASC -se toma este valor por defecto) ó descendente (DESC)

```
clima=# SELECT nombre,x,y,z
FROM observatorios
ORDER BY x,y
DESC;
```

4.3 Consultas con Predicado

Una manera de limitar el número de filas que devuelve el servidor es utilizar predicados en la selección. El predicado se incluye entre la claúsula y el primer nombre del campo a recuperar, los posibles predicados son:

***** Devuelve todos los campos de la tabla. En este caso el servidor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL.

```
clima=# SELECT *
FROM observatorios;
```

No es conveniente abusar de este predicado ya que obligamos al servidor a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
clima=# SELECT indentinm,x,y,z,nombre
FROM observatorios;
```

DISTINCT Omite los registros cuyos campos seleccionados coincidan totalmente. Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente.

```
clima=# SELECT DISTINCT indentinm,x,y,z,nombre
FROM observatorios;
```

DISTINC ON (*campo***)** Omite registros que coincidan en el campo seleccionado. Por ejemplo la siguiente orden devuelve un sólo observatorio por valor de altitud:

```
clima=# SELECT DISTINCT ON (z) indentinm,x,y,z,nombre
FROM observatorios;
```

4.4 Alias

En determinadas circunstancias es necesario asignar un nuevo nombre a alguna de las columnas devueltas por el servidor. Para ello tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada:

clima=# SELECT nombre,x AS longitud, y AS latitud, z AS altitud
FROM observatorios;

5 Criterios de Selección

En la sección anterior se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla, salvo que se untilizara el predicado **DISTINCT**. En esta sección se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan una condiciones preestablecidas.

5.1 La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT.

Por ejemplo, para obtener sólo los observatorios situados a más de 500 metros de altitud, la consulta adecuada sería:

clima=# SELECT nombre,x,y,z
FROM observatorios
WHERE z > 500;

5.2 Operadores Lógicos

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis:

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

- Falso AND Verdad Falso
- Falso AND Falso Falso
- Verdad OR Falso Verdad
- Verdad OR Verdad Verdad
- Falso OR Verdad Verdad

- Falso OR Falso Falso

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

clima=# SELECT nombre,x,y,z
FROM observatorios
WHERE x > 600000 AND x < 650000;

clima=# SELECT nombre,x,y,z
FROM observatorios
WHERE (x > 600000 AND x < 650000) OR z<200;

La última consulta devolvera los observatorios situados entre los valores de X UTM de 600000 y 650000 UTM y aquellos con altitud inferior a 200 metros.

5.3 Intervalos de Valores

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

campo **[Not] Between** *valor1* **And** *valor2*

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo:

clima=# SELECT nombre,x,y,z
FROM observatorios
WHERE x Between 600000 AND 650000;

esta orden es equivalente a **bd=# SELECT nombre,x,y,z**
FROM observatorios WHERE x > 600000 AND Edad < 650000;

5.4 El Operador

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión modelo

En donde expresión es una variable y modelo un patrón de texto con el que se compara la expresión. Se puede utilizar este operador para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Lorca), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	’a*a’	’aa’, ’aBa’, ’aBBBa’	’aBC’
Carácter especial	’a[*]a’	’a*a’	’aaa’
Varios caracteres	’ab*’	’abcdefg’, ’abc’	’cab’, ’aab’
Un solo carácter	’a?a’	’aaa’, ’a3a’, ’aBa’	’aBBBa’
Un solo dígito	’a#a’	’a0a’, ’a1a’, ’a2a’	’aaa’, ’a10a’
Rango de caracteres	’[a-z]’	’f’, ’p’, ’j’	’2’, ’&’
Fuera de un rango	’[!a-z]’	’9’, ’&’, ’%’	’b’, ’a’
Distinto de un dígito	’[!0-9]’	’A’, ’a’, ’&’, ’ ’	’0’, ’1’, ’9’
Combinada	’a[!b-m]#’	’An9’, ’az0’, ’a99’	’abc’, ’aj0’

Table 1: Posibilidades del operador Like

El operador se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduces **C*** en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

La tabla 1 muestra cómo utilizar el operador para comprobar expresiones con diferentes modelos.

El siguiente ejemplo devolvería todos los observatorios en cuyo nombre apareciera incluida la palabra Lorca:

clima=# SELECT * from observatorios
WHERE nombre ’Lorca’;

5.5 El Operador In

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

expresión **[Not] In(***valor1, valor2, . . .***)**

clima=# SELECT *
FROM observatorios
WHERE indentinm IN(7149,7069);

6 Agrupamiento de Registros

6.1 GROUP BY y HAVING

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo

GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no se desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.

Como ejemplo vamos a hacer la primera consulta a una nueva tabla :

clima=# SELECT *
FROM menstem;

Esta tabla contiene los valores de temperatura mensual para algunos de los observatorios incluidos en la base de datos. Si a partir de esta tabla quisieramos conocer las temperaturas medias mensuales en el observatorio de *Aguilas Montagro* (identificador 7001E):

clima=# SELECT mes,AVG(tmed)
FROM menstem
WHERE ide='7001E'
GROUP BY mes;

si ademas añadimos lo siguiente:

clima=# SELECT mes,AVG(tmed),COUNT(tmed)
FROM menstem
WHERE ide='7001E'
GROUP BY mes;

Tendremos no sólo las medias sino también el número de años utilizado para calcular estas medias.

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

clima=# SELECT mes,AVG(tmed),COUNT(tmed)
FROM menstem
WHERE ide='7001E'
GROUP BY mes
HAVING AVG(tmed)>20;

6.2 AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente

Avg(*expr***)**

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por Avg es la media aritmética (la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

clima=# SELECT Avg(tmed) AS Promedio
FROM menstem
WHERE ide='7001E';

6.3 Count

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente

Count(*expr***)**

En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe poner el asterisco entre dobles comillas ('*').

clima=# SELECT Count(*) AS Total
FROM observatorios;

Si expr identifica a múltiples campos, la función Count cuenta un registro sólo si al menos uno de los campos no es Null. Si todos los campos especificados son Null, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

clima=# SELECT Count(x & y & z) AS Total
FROM observatorios;

6.4 Max, Min

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:

Min(*expr***)**
Max(*expr***)**

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL.

clima=# SELECT Min(Gastos) AS ElMin
FROM Pedidos
WHERE Pais = 'España';

clima=# SELECT Max(Gastos) AS ElMax
FROM Pedidos
WHERE Pais = 'España';

6.5 Stddev

Devuelve estimaciones de la desviación estándar para la población (el total de los registros de la tabla) o una muestra de la población representada (muestra aleatoria) . Su sintaxis es:

STDDEV(*expr***)**

En donde expr representa el nombre del campo que contiene los datos que desean evaluarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL)

StDevP evalúa una población, y StDev evalúa una muestra de la población. Si la consulta contiene menos de dos registros devuelve un valor Null (el cual indica que la desviación estándar no puede calcularse). La siguiente consulta:

clima=# SELECT mes,Stddev(tmed) AS desviacion,count(tmed) AS datos
FROM menstem
WHERE ide='70001E'
GROUP BY mes;

devolverá la desviación típica y el tamaño muestral de la precipitación mensual en el observatorio 70001E (Aguilas Montagro).

6.6 Sum

Devuelve la suma del conjunto de valores contenido en un campo especifico de una consulta. Su sintaxis es:

Sum(*expr***)**

En donde expr respresenta el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). En el siguiente ejemplo vamos a utilizar la tabla **menspluv** que contiene datos de precipitación mensual para obtener una tabla de precipitación anual.

```
clima=# SELECT ano,sum(pluv) AS precipitacion, count(pluv) as meses from menspluv
WHERE pluv>=0 AND ide='7094' GROUP BY ano;
```

Date cuenta de que creamos una nueva variable denominada meses que almacena el número de meses utilizados para calcular la suma y sirve para verificar que el año este completo en la tabla.

6.7 Limitar el número de registros devueltos por el servidor

Finalmente, puede limitarse el número de registros devueltos por la orden **SELECT** utilizando el modificador **LIMIT**. Su sintaxis sería:

SELECT FROM *tabla* **LIMIT** *número*

lo que devolvería sólo *número* registros. Resulta útil para consultas del tipo ¿Cuales son los 5 observatorios situados a mayor altitd?

```
clima=# SELECT nombre,z
FROM observatorios
ORDER BY z DESC
LIMIT 5;
```

6.8 Consultas a varias tablas

Pueden combinarse varias tablas mediante operaciones de SQL más complejas. Cuando se combinan tablas suele introducirse un *alias* simplificado para las tablas (la primetra letra por ejemplo). En el siguiente ejemplo se va a utilizar, además de la tabla **observatorios**, una nueva tabla llamada **menstem** que contiene datos acerca de la temperatura mensual en los observatorios utilizados. Puedes ver los contenidos de esta tabla con la orden

menstem y comprobar que existe una columna denominada **ide** que actúa como campo común con la columna **indentinm** de la tabla **observatorios**.

Vamos a obtener una tabla que nos podría servir para interpolar la temperatura media del mes de Julio de 1990. Los primero que vamos a hacer es ver con cuantos datos dispondríamos para la intrpolación:

```
bd=#SELECT *
WHERE ano=1990 and mes=7;
```

A continuación haremos una consulta que combine la tabla **observatorios** con la tabla **menstem** para obtener una tabla con la que poder hacer la interpolación:

```
clima=#SELECT x,y,z,tmed
FROM observatorios, menstem
WHERE ide=indentinm and mes=7 and ano=1990;
```

Como ves no hay grandes diferencias, es necesario declarar las dos tablas tras el modificador WHERE y especificar que el identificador de los observatorios, que actúa en este caso como campo común, debe ser igual para combinar los registros. Este es un caso especialmente simple porque no hay nombres de columna repetidos en ambas tablas, en caso de que esto hubiera sido así deberíamos haber utilizado la sintaxis:

```
clima=#SELECT o.x,o.y,o.z,t.tmed
FROM observatorios o, menstem t
WHERE t.ide=o.indentinm and mes=7 and ano=1990;
```

donde especificamos a que tabla pertenece cada nombre de columna (tabla.columna) utilizando un *alias* en lugar de los nombres de las tablas (o y t) para abreviar.

7 subconsultas

Una subconsulta es una instrucción SELECT escrita ente paréntesis y anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta. Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING.

En una subconsulta, se utiliza la instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING de la consulta principal. Por ejemplo la siguiente consulta utiliza una subconsulta para calcular la altitud media de los observatorios y devolver aquellos observatorios situads a mayor altitud:

```
clima=# SELECT *
FROM observatorios
WHERE z > (
SELECT AVG(z)
```

FROM observatorios
);

En este caso la comparación se realiza con un operador simple (mayor que) ya que la subconsulta devuelve un sólo valor, en el caso de que la subconsulta devuelva varios valores es necesario utilizar predicados de comparación más sofisticados como ANY, ALL, IN o EXISTS.

7.1 ANY

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento.:

clima=# SELECT *
FROM observatorios
WHERE z > ANY (
SELECT *
FROM observatorios
WHERE nombre 'Lorca'
);

Devolvera aquellos observatorios cuya altitud sea mayor que la de **cualquiera de** los observatorios que contienen "Lorca" en su nombre, es decir devolverá los situados a mayor altura que el más bajo de estos.

7.2 ALL

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

clima=# SELECT *
FROM observatorios
WHERE z < ALL (
SELECT *
FROM observatorios
WHERE nombre 'Lorca'
);

Devolverá aquellos observatorios cuya altitud sea mayor que la de **todos** los observatorios que contienen "Lorca" en su nombre, es decir devolverá los situados a mayor altura que el más alto de estos.

7.3 IN

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual.

```
clima=# SELECT *
FROM observatorios
WHERE z IN (
SELECT *
FROM observatorios
WHERE nombre 'Lorca'
);
```

devolvera aquellos observatorios cuya altitud sea igual a la altitud de algunos de los observatorios que incluyen la palabra 'Lorca' en su numbre.

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

7.4 EXISTS

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

```
clima=# SELECT *
FROM observatorios
WHERE EXISTS (
SELECT z
FROM observatorios
WHERE nombre 'Lorca');
```

Devolverá todos los registros porque en la base de datos existen observatorios cuyo nombre incluye la palabra 'Lorca'.

```
clima=# SELECT *
FROM observatorios
WHERE EXISTS (
SELECT z
FROM observatorios
WHERE nombre 'Pamplona');
```

No devolverá ningún registro porque en la base de datos no existen observatorios cuyo nombre incluya la palabra 'Pamplona'.

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta.

8 Consultas de Acción

Las consultas de acción son aquellas que no devuelven ningún registro, son las encargadas de acciones como añadir y borrar y modificar registros.

8.1 DELETE

Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Su sintaxis es:

DELETE FROM *Tabla* **WHERE** *criterio*

Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacerse la operación. Si quiere saber qué registros se eliminarán, primero examina los resultados de una consulta de selección que utilice el mismo criterio y después ejecuta la consulta de borrado. En todo caso es conveniente tener copias de seguridad de las tablas involucradas en una consulta de eliminación. Si se eliminan los registros equivocados podrás recuperarlos desde las copias de seguridad.

```
clima=# DELETE
FROM observatorios
WHERE x <1000;
```

8.2 INSERT INTO

Agrega un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipo: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla.

8.2.1 Para insertar un único Registro:

En este caso la sintaxis es la siguiente:

```
bd=# INSERT INTO Tabla (campo1, campo2, .., campoN)
VALUES (valor1, valor2, ..., valorN);
```

Esta consulta guarda en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples (’) los valores literales (cadenas de caracteres).

8.2.2 Para insertar Registros de otra Tabla:

En este caso la sintaxis es:

bd=# INSERT INTO Tabla (campo1, campo2, ..., campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN
FROM TablaOrigen;

En este caso se seleccionarán los campos 1,2, ..., n dela tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición SELECT puede incluir la cláusula WHERE para filtrar los registros a copiar. Si Tabla y TablaOrigen poseen la misma estrucutra podemos simplificar la sintaxis a:

bd=#INSERT INTO Tabla
SELECT TablaOrigen.*
FROM TablaOrigen;

De esta forma los campos de TablaOrigen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de TablaOrigen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementandose como le corresponde.

Se puede utilizar la instrucción INSERT INTO para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código específica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar INSERT INTO para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula SELECT ... FROM como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula SELECT especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta.

Si la tabla destino contiene una clave principal, hay que segurarse que es única, y con valores no-Null ; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador , no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula IN para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula VALUES. Si se omite la lista de campos, la cláusula VALUES debe incluir un valor para cada campo de la tabla, de otra forma fallará INSERT.

clima=#INSERT INTO observatorios (indentinm,nombre, x,y,z,ident,obs)
VALUES (’9999’, ’Murcia inventado’,650000,4200000„450);

8.3 UPDATE

Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico. Su sintaxis es:

bd=#UPDATE Tabla
SET Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN
WHERE Criterio;

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez:

clima=#UPDATE observatorios
SET indentinm = 8888
WHERE obs = 450;

UPDATE no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

clima=#SELECT *
WHERE obs=450; clima=#UPDATE observatorios
SET indentinm = 8888
WHERE obs = 450;

Si en una consulta de actualización suprimimos la cláusula WHERE todos los registros de la tabla señalada serán actualizados, por lo que hay que tener precaución con este tipo de consulta.

clima=#UPDATE observatorios
SET nombre = ’te has cargado la tabla’;

9 R como cliente de PostgreSQL

El programa **psql** es simplemente uno de los muchos clientes de bases de datos disponibles. Existe incluso la posibilidad de convertir cualquier programa en un cliente de base de datos añadiendole las funciones para lanza consultas al servidor e interpretar las respuestas de este.

Se han desarrollado multitud de paquetes que amplian las capacidades del programa R. Uno de estos es **RPgSQL** que incorpora funciones para conectar con PostgreSQL. Una típica sesión de trabajo sería:

```
1> library(RPgSQL)
2> db.connect(dbname="clima",user=”usuario”)
3> orden="select indentinm,x,y,z from observatorios;"
4> db.execute(orden,clear=F)
5> datos<-db.fetch.result()
6> db.clear.result()
```

1. Carga el paquete **RPgSQL** dando acceso a sus funciones;
2. Conecta a la base de datos (en algunos casos será necesario pasar una contraseña);
3. Almacena el texto de una consulta en una variable;
4. Lanza la consulta al servidor;
5. Almacena los resultados en una **data.frame** llamado *datos*;
6. Borra de la memoria los resultados de la consulta (esta última no es necesaria pero sirve para liberar memoria del ordenador).

Una vez que se dispone de los resultados de la consulta en un **data.frame**, podemos utilizar cualquiera de las funciones de R para analizar los resultados:

```
>plot(datos$x,datos$y)
```