UNIVERSIDAD DE GUANAJUATO

Lenguajes para DBMS
Ene-Jun 2020

Dr. Andrés Espinal Jiménez


Harrington, J. L. (2010).
SQL Clearly Explained.
The Morgan Kaufmann Series in Data Management Systems, 3 edition.


Stones, R. & Matthew, N. (2006).
Beginning Databases with PostgreSQL: From Novice to Professional.
Apress.

Documentación oficial PostgreSQL:
`https://www.postgresql.org/docs/11/index.html`

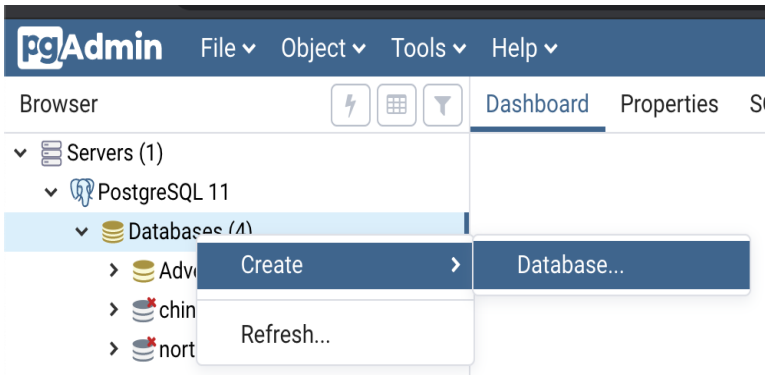
Introducción a los Sistemas de Administración de Bases de Datos (DBMS)

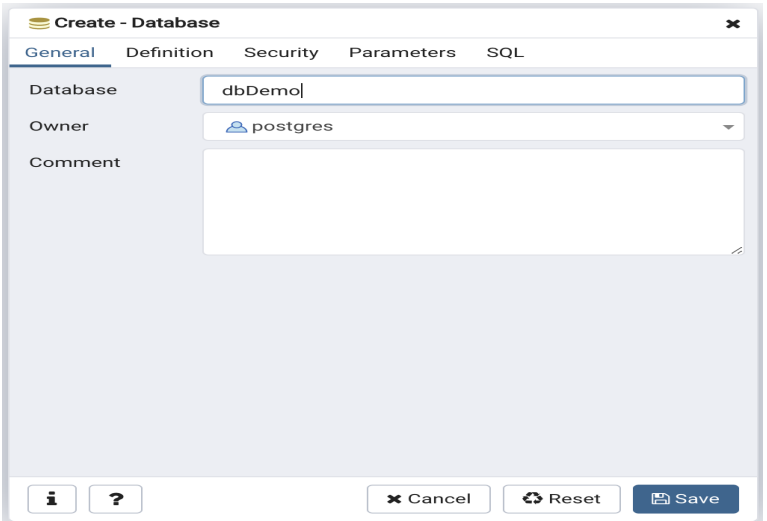
Lenguaje de Definición de Datos

Las sentencias SQL para crear y borrar una base de datos se muestran en el siguiente bloque de código en la línea *1* y *2* respectivamente.

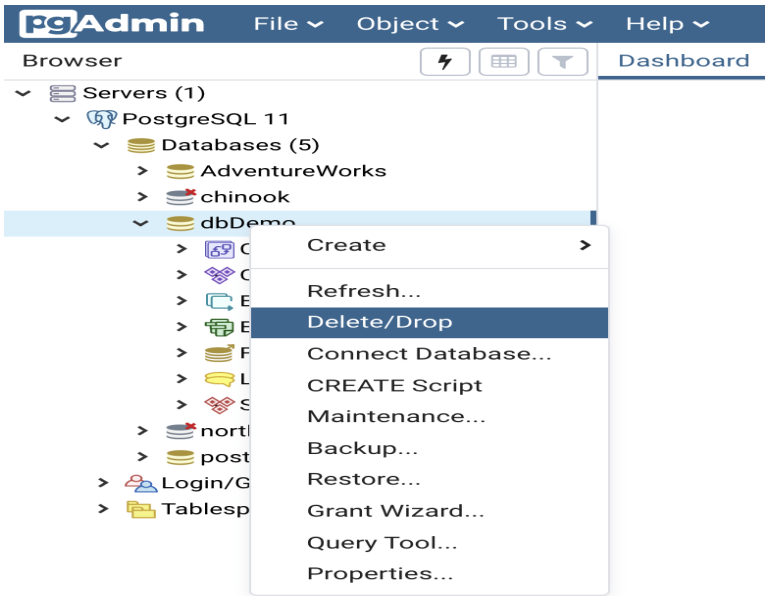
```
1 CREATE DATABASE basedatos_nombre;
2 DROP DATABASE basedatos_nombre;
```

De manera análoga, esto se puede lograr a través del pgAdmin. A continuación se muestra el procedimiento para crear una base de datos en pgAdmin.





A continuación se muestra el procedimiento para borrar una base de datos en pgAdmin.



Las sentencias SQL para crear una tabla se muestran en el siguiente bloque de código.

```
1  CREATE  [TEMPORARY] TABLE nombre_tabla(
2  {nombre_columna dtype [column-constraint]
3  [,...]}
4  [CONSTRAINT table-constraint]);
```

- En la línea *1*, la parte **[TEMPORARY]** es opcional, agregando esta instrucción la tabla no se almacena permanentemente en la base de datos y se borra al cerrar sesión. El nombre de la tabla (**nombre_tabla**) debe ser único en la base de datos.

- Las lineas *2* y *3*, denotan el conjunto de columnas de la tabla (señalado entre **{}**, estos símbolos deben ser omitidos), cada atributo se le asigna un nombre único (**nombre_columna**) en la tabla, un tipo de dato (**dtype**) y una o varias restricciones de columnas (**[column-constraint]**, las cuales son opcionales). El símbolo **[,...]** denota la separación de columnas por una coma en caso de que la tabla contenga más de una columna.
- En **[CONSTRAINT table-constraint]** se especifícan las restricciones a nivel de tabla; estas son opcionales.

Los tipos de datos básicos en un adminstrador de bases de datos relacionales son:

- Boolean
- Character
- Number (Integer y Floating-point)
- Temporal

En particular PostgreSQL maneja más tipos de datos, los cuales se pueden consultar en:

`https://www.postgresql.org/docs/current/datatype.html`

Los tipos de datos booleanos (**BOOLEAN** ó simplemente **BOOL**) pueden almacenar dos posibles valores *true* y *false*.

Para este tipo de dato, en PostgreSQL, existen diversos valores que pueden ser interpretados como true o false. Estos valores se muestran en la siguiente tabla:

True	False
'1'	'0'
'yes'	'no'
'y'	'n'
'true'	'false'
't'	'f'

Principalmente hay tres variantes del tipo de dato: un solo carácter, una cadena de caracteres de longitud fija y cadenas de caracteres de logintud variable. La siguiente tabla muestra la descripción de cada uno de ellos:

Definición	**Descripción**
CHAR	Un solo carácter.
CHAR(n)	Un conjunto de exactamente *n* caracteres.
VARCHAR(n)	Un conjunto de hasta *n* caracteres.

PostgreSQL permite no especificar *n* en el tipo de dato VARCHAR lo que resulta en una cadena de caracteres de logintud "ilimitada". Esto también se logra con el tipo de dato **TEXT**, sin necesidad de obviar una longitud; una extensión de PostgreSQL del estándar SQL.

Básicamente hay dos variantes del tipo de dato. La siguiente tabla muestra su descripción:

Definición	Descripción
SMALLINT	Entero de 2 bytes con signo. Puede almacenar números de -32768 a 32767.
INTEGER (ó **INT**)	Entero de 4 bytes con signo. Puede almacenar números de -2147483648 a 2147483647.

Existe una extesión del estándar SQL que define un tipo de dato llamado **SERIAL** similar a INTEGER con la diferencia que su valor es normalmente automáticamente llenado por PostgreSQL (autoincremental).

Existen dos variantes del tipo de dato: de punto flotante de propósito general y con precisión fija. Estos se describen a continuación:

Definición	**Descripción**
FLOAT(n)	Punto flotante con la precisión de n. Almacena hasta 8 bytes (Inexacto).
NUMERIC(p,s)	Real con p dígitos con s de ellos después del punto decimal (Exacto).

Existen cuatro variantes del tipo de dato para describir diferentes unidades de tiempo. Estos se describen a continuación:

Definición	Descripción
DATE	Almacena fecha
TIME	Almacena hora
TIMESTAMP	Almacena fecha y hora
INTERVAL	Almacena información sobre diferencia en unidades temporales

Las restricciones a nivel de columna permiten imponer reglas conocidas que gobiernen los datos. A continuación se muestran las restricciones a nivel de columna:

Restricción	Significado
NOT NULL	La columna no puede tener almacenado un valor NULL.
UNIQUE	El valor almacenado en la columna debe ser diferente para cada renglón.
PRIMARY KEY	Una combinación de NOT NULL y UNIQUE. Cada tabla solo puede tener una columna con esta restricción.
DEFAULT *valor-defecto*	Permite asignar un valor por defecto cuando se insertan datos.
CHECK *condicion*	Permite verificar una condición cuando se insertan o actualizan datos.
REFERENCES	Restringe el valor a ser uno que exista en una columna de una tabla diferente.

Estas restricciones son similares a las de nivel de columna. Permiten aplicar restricciones a la tabla entera en lugar de una sola columna; como por ejemplo definir una llave primaria compuesta. A continuación se muestran las restricciones a nivel de tabla:

Restricción	**Significado**
UNIQUE(*lst-cols*)	El valor almacenado en las columnas listadas debe ser diferente para todas en cada renglón.
PRIMARY KEY(*lst-cols*)	Una combinación de NOT NULL y UNIQUE. Cada tabla puede tener solo una llave primaria.
CHECK *condicion*	Permite verificar una condición cuando se insertan o actualizan datos.
REFERENCES	Restringe el valor a ser uno que exista en una columna de una tabla diferente.

A una tabla se le pueden agregar, eliminar y renombrar columnas, incluso cambiar su definición de tipos de datos. También es posible renombrar la tabla en sí. Todas estas operaciones pueden ser realizadas mientras la data contiene datos. Las instrucciones para estas operaciones son listadas a continuación.

```
1  ALTER TABLE nombre_tabla
2          ADD COLUMN nombre_columna dtype
3  ALTER TABLE nombre_tabla
4          DROP COLUMN nombre_columna
```

- La instrucción en las lineas *1* y *2* permite agregar una columna junto con su tipo de dato a una tabla existente.
- La instrucción en las lineas *3* y *4* elimina la columna nombrada de una tabla.

UNIVERSIDAD DE GUANAJUATO

```
1  ALTER TABLE nombre_tabla
2          RENAME COLUMN nombre_columna
3          TO nombre_columna_nuevo
4  ALTER TABLE nombre_tabla
5          ALTER COLUMN nombre_columna
6          TYPE dtype_nuevo [USING expresion]
```

- La instrucción en las lineas *1* a *3* es para renombrar una columna existente.
- La instrucción en las lineas *4* a *6* se usa para cambiar el tipo de dato de una **[USING expresion]** sirve para definir un *cast* explícito.

UNIVERSIDAD DE GUANAJUATO

```
1  ALTER TABLE nombre_tabla
2          ALTER COLUMN nombre_columna
3          [SET DEFAULT valor | DROP DEFAULT]
4  ALTER TABLE nombre_tabla
5          ALTER COLUMN nombre_columna
6          [SET NOT NULL | DROP NOT NULL]
```

- La instrucción en las lineas *1* a *3* permite colocar (**[SET DEFAULT valor]**) o eliminar (**[DROP DEFAULT]**) un valor por defecto a la columna referida.
- La instrucción en las lineas *4* a *6* sirve para permitir (**[DROP NOT NULL**) o no (**[SET NOT NULL]**) valores nulos en una columna.

```
1  ALTER TABLE nombre_tabla
2          ADD CHECK (condicion)
3  ALTER TABLE nombre_tabla
4          ADD CONSTRAINT nombre
5          definicion-restriccion
```

- La instrucción en las lineas *1* y *2* permite agregar una verficicación en la tabla.
- La instrucción en las lineas *3* y *4* es para definir una restricción a la tabla.

```
1  ALTER TABLE nombre_tabla
2          DROP CONSTRAINT nombre
3  ALTER TABLE nombre_tabla
4          RENAME TO nombre_tabla_nuevo
```

- La instrucción en las lineas *1* a *3* es para eliminar una restricción en la tabla.
- La instrucción en las lineas *4* a *6* se usa para cambiar el nombre de una tabla.

UNIVERSIDAD DE GUANAJUATO

```
1  INSERT INTO nombre_tabla
2          VALUES (valor_atributo1,
3          ...,valor_atributoN)
4  INSERT INTO nombre_tabla
5          (atributoX[,...])
6          VALUES (valor_atributoX[,...])
```

- La instrucción en las lineas *1* a *3* sirve para insertar un registro incluyendo un valor para cada atributo de la tabla.
- La instrucción en las lineas *4* a *6* inserta un registro especificando explícitamente una lista de atributos y sus respectivos valores.

Lenguaje de Manipulación de Datos

La recuperación de datos en SQL es realizada a través de la sentencia **SELECT**. Con ella se pueden elegir columnas y filas a recuperar de una tabla o la combinación de varias tablas, agrupar datos y desarrollar cálculos sencillos. Su forma básica es la siguiente:

```
1  SELECT [DISTINCT][*|nombre_columnaX[,...]]
2  FROM nombre_tabla
```

- En la línea *1*, el símbolo * sirve para recuperar todas las columnas de la tabla a consultar; este puede ser sustituido por una lista de uno o más atributos especificados explícitamente (**nombre_columna[,...]**). La palabra opcional **DISTINCT** permite remover filas repetidas.
- La línea *2* permite especificar que tabla será consultada (**nombre_tabla**).

Los resultados de una sentencia SELECT pueden ser ordenados con base los valores de una o varias columnas, cada una de ellas en orden ascendente o descendente. La sintaxis es como sigue:

```
1  SELECT [DISTINCT][*|nombre_columnaX[,...]]
2  FROM nombre_tabla
3  ORDER BY nombre_columnaX [ASC|DESC][,...]
```

- En la línea *3*, la parte **[ASC|DESC]** puede ser omitida y el ordenamiento por defecto será ascendente. Sin embargo, está debe ser explícitamente definida u omitida para cada columna que estará involucrada en el ordenamiento.

Como anteriormente se mencionó, la sentencia SELECT puede ser usada para seleccionar renglones que cumplan uno o varios criterios especificados en un predicado. La sintaxis es la siguiente:

```
1  SELECT [DISTINCT][*|nombre_columnaX[,...]]
2  FROM nombre_tabla
3  WHERE predicado
```

- En la línea *3* se debe definir un predicado para discriminar filas. En este predicado se pueden usar operadores relacionales, lógicos, paréntesis para precedencia y operadores especiales.

Los operadores relacionales son usados para expresar relaciones. Estos son mostrados a continuación

Operador	Significado
=	Igual a
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
!=	No igual a

Los operadores lógicos sirven para concatenar predicados. Esto sirve cuando una sola expresión lógica no es suficiente para identificar las filas que se quieren recuperar

Operador	Significado
AND	Y, ambas expresiones lógicas deben ser verdaderas para dar un valor de verdadero
OR	O, alguna o ambas expresiones lógicas deben ser verdaderas para dar un valor de verdadero
NOT	No, niega una expresión lógica

- Operador **BETWEEN**. Simplifíca la escritura de predicados que buscan valores en un intervalo inclusivo de valores. Para discriminar filas en un intervalo de valores para una columna, el operador BETWEEN se usaría: edad BETEWEEN 10 AND 18, en lugar de usar: edad >= 10 AND edad <= 18.
- Operador **IN**. Compara los valores de una columna contra un conjunto de valores. El operador regresa verdadero si el valor está dentro del conjunto. Para discriminar filas con respecto a un conjunto de valores para una columna, el operador IN se usaría: edad IN (10,12,14,16), en lugar de usar: edad = 10 OR edad = 12 OR edad = 14 OR edad = 16.

- Operador **LIKE**. Permite encontrar patrones o subcadenas en cadenas de caracteres mediante el uso de marcadores de posición. Con el símbolo **%** se indican la ocurrencia de cero o más caracteres y con _ se indica la presencia de un carácter. Por ejemplo, para encontrar todos los todos los nombres que comiencen con *'Sam'*, el operador LIKE se usaría: nombre LIKE 'Sam%', esto regresaría nombres como *'Samantha'* ó *'Samuel'*. O para recuperar los nombres como *'Cony'* ó *'Tony'*, el operador debería usarse como nombre LIKE '_ony'.
- Operador **IS NULL**. Permite identificar valores nulos en un predicado con el objetivo de recuperar renglones que contienen este *indicador*. Un ejemplo de su uso es: vigente IS NULL.


SOMOS
UG