Arquitectura y Administración de bases de datos con SQL 2021

Héctor Manuel Garduño Castañeda

Diciembre, 2021

scidata

Contenido

Creación de bases de datos

Insertar registros

Importado de tablas

Selecciones

Operadores lógicos

Actualizaciones

Borrado

Cambiar estructura de la tabla

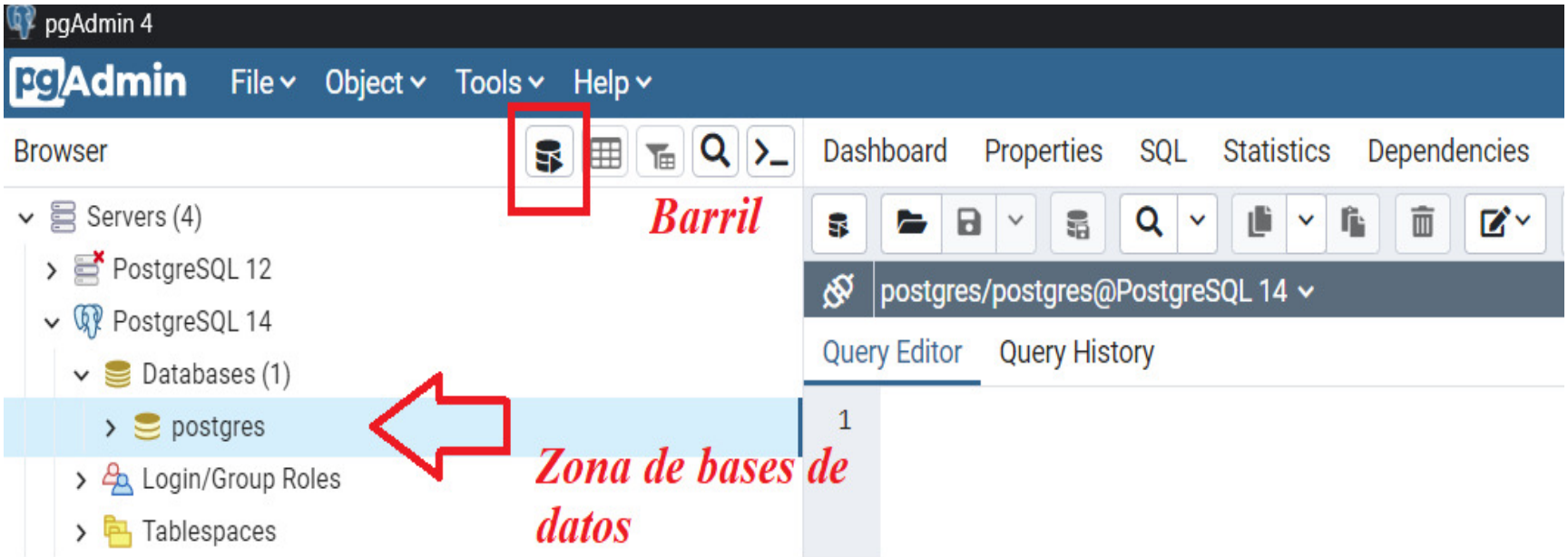
¿Qué son las bases de datos?

Una base de datos es un *almacén* que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente.

Se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada. Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas, también llamadas campos, guardan una parte de la información sobre cada elemento que queramos guardar en la tabla. Cada fila de la tabla conforma un registro.

Los scripts en SQL

Al tratarse de un lenguaje de programación, PostgreSQL, mediante el software pgAdmin 4, nos permite escribir el lenguaje en scripts, que son editores de escritura. En este caso, para abrir un script nuevo, simplemente nos colocamos en alguna de las bases de datos que ya existan y damos click en el botón de *barril*.



Creación de una base de datos

pgAdmin 4 tiene por default una base de datos llamada postgres, por lo que siempre podemos abrir un script nuevo. Para crear una base de datos nueva, se puede hacer con click secundario en Database, o en un script con el comando CREATE DATABASE ⟨⟨nombre de la base de datos⟩⟩. A su vez, para eliminar una base de datos escribimos DROP DATABASE ⟨⟨nombre de la base de datos⟩⟩.

Importante. SQL no hace distinción entre mayúsculas y minúsculas en su sintaxis. Por ejemplo, pudimos haber escrito **create database hola** para crear una base de datos de nombre *hola*. Sin embargo, es preferible usar mayúsculas para darle más claridad al script cuando escribimos comandos. A su vez, es recomendable siempre terminar con **;** cada línea de código para indicar que se ha terminado una línea.

Ejemplo. Crear una base de datos de nombre *Entrenamiento*:
CREATE DATABASE Entrenamiento;

Práctica

Crear una base de datos llamada *Informacion* (**tip:** evita el uso de acentos en lenguajes de programación).

CREATE DATABASE Informacion;

Creación de tablas en una base de datos

Para crear una tabla en una base de datos, lo podemos hacer desde el script con las siguientes instrucciones:

CREATE TABLE ⟨⟨**nombre**⟩⟩**(**
⟨⟨nombre columna 1⟩⟩ ⟨⟨tipo de datos⟩⟩ ⟨⟨[condiciones]⟩⟩,
⟨⟨nombre columna 2⟩⟩ ⟨⟨tipo de datos⟩⟩ ⟨⟨[condiciones]⟩⟩,
...
⟨⟨nombre columna n⟩⟩ ⟨⟨tipo de datos⟩⟩ ⟨⟨[condiciones]⟩⟩**);**

Condiciones de las columnas

Algunas de las condiciones fundamentales que pueden tener las columnas son **NOT NULL**, **UNIQUE** y **PRIMARY KEY**.

NOT NULL indica que ningún valor de esa columna puede ser vacío.
UNIQUE indica que ningún valor de esa columna puede repetirse.
PRIMARY KEY indica que el valor de esa columna sirve para identificar completamente cada registro.

Ejemplo.

```
CREATE TABLE prueba(
    col1 int PRIMARY KEY,
    col2 char(2) NOT NULL,
    col3 char,
    col4 int UNIQUE
);
```

Práctica

En la base de datos Información, crear una tabla llamada tabla_clientes con 5 columnas: Id_cliente, Nombre, Apellido, Edad y correo.

```
CREATE TABLE tabla_clientes(
    Id_cliente int,
    Nombre varchar,
    Apellido varchar,
    Edad int,
    correo varchar
);
```

Una vez que ya sabemos crear bases de datos y tablas, corresponde aprender a añadir los registros de cada tabla. Para ello usamos el comando **INSERT INTO** que tiene la siguiente sintaxis general:

INSERT INTO ⟨⟨nombre de la tabla⟩⟩**(**⟨⟨nombre de columna 1⟩⟩,⟨⟨nombre de columna 2⟩⟩**,...)**
VALUES (⟨⟨Valor 1⟩⟩,⟨⟨Valor 2⟩⟩**);**

La sintaxis anterior permite añadir un único renglón a la tabla, pero puede ser modificada de diversas maneras. Sin embargo, lo más importante es tener presente que se deben cumplir los tipos de valores que las columnas aceptan y las condiciones.

Errores típicos

En la tabla *prueba*, ejecutar los siguientes códigos y comprender por qué falla cada uno de ellos.

INSERT INTO prueba(col1)
VALUES (2);

INSERT INTO prueba(col1, col2)
VALUES (2,'abc');

INSERT INTO prueba(col1, col2)
VALUES (2,'ab','abc');

INSERT INTO prueba(col1, col2, col3)
VALUES (2,'ab','a');

INSERT INTO prueba(col1, col2, col3)
VALUES (2,'c','3');

Práctica

Cuando conoces el orden de las columnas de una tabla, sus tipos de datos y las condiciones, es posible modificar la sintaxis de INSERT para incluso añadir mas de un registro. Por ejemplo, en la tabla *tabla_clientes*:

Añadir un solo registro
INSERT INTO tabla_clientes
VALUES (1, 'Cintia', 'Cee', 32, 'ab@xyz.com');

Añadir un solo registro especificando las columnas
INSERT INTO tabla_clientes(Id_cliente, Nombre, Edad, correo)
VALUES (2, 'Diana', 22, 'd@xyz.com');

Añadir varios registros
INSERT INTO tabla_clientes
VALUES (3, 'Euclides', 'Ef', 27, 'ef@xyz.com'),
(4, 'Gabriela', 'Eh', 35, 'gh@xyz.com');

Muchas veces las tablas ya han sido previamente construidas en un archivo de tipo csv o block de notas y requerimos leerlas en SQL. Para ello tenemos el comando **COPY** cuya sintaxis general es

COPY ⟨⟨nombre de la tabla⟩⟩(⟨⟨nombre de la columna 1⟩⟩, ⟨⟨nombre de la columna 2⟩⟩,...)
FROM 'dirección y nombre del archivo.csv' **DELIMITER** ',' **CSV HEADER;**

Práctica

Leer los archivos copy.csv y copytext.txt, de la sección data del repositorio, para añadir ambas tablas a la tabla *tabla_clientes*.

COPY tabla_clientes(Id_cliente, Nombre, Apellido, Edad, correo)
FROM 'C:\Users\Public\Documents\aabd_sql_2021\copy.csv'
DELIMITER ',' **CSV HEADER;**

COPY tabla_clientes(Id_cliente, Nombre, Apellido, Edad, correo)
FROM 'C:\Users\Public\Documents\aabd_sql_2021\copytext.txt'
DELIMITER ',' **;**

La instrucción **SELECT** se utiliza para obtener los datos de una tabla dentro de una base de datos y que devuelva la selección en forma de tabla. Estas tablas de resultados se denominan conjuntos de resultados.

SELECT ⟨⟨nombre de alguna columna⟩⟩, ⟨⟨nombre de alguna columna⟩⟩,...**FROM**⟨⟨nombre de la tabla⟩⟩**;**

SELECT ∗ **FROM** ⟨⟨nombre de la tabla⟩⟩**;**

Práctica

De la tabla *tabla_clientes* mostrar la columna de nombres; mostrar nombres y apellidos; mostrar todas las columnas.

Seleccionar solo una columna
SELECT Nombre **FROM** tabla_clientes**;**

Seleccionar dos columnas
SELECT Nombre, Apellido **FROM** tabla_clientes**;**

Seleccionar todas las columnas
SELECT $*$ **FROM** tabla_clientes**;**

La palabra clave **DISTINCT** sirve para, en conjunción con **SELECT**, mostrar valores sin duplicados y elegir valores únicos.

SELECT DISTINCT ⟨⟨nombre de alguna columna⟩⟩, ⟨⟨nombre de alguna columna⟩⟩,...**FROM**⟨⟨nombre de la tabla⟩⟩**;**

Práctica

De la tabla *tabla_clientes* mostrar la columna de nombres sin duplicados; mostrar nombres y edades sin duplicados.

Seleccionar solo una columna
SELECT DISTINCT Nombre **FROM** tabla_clientes**;**

Seleccionar dos columnas
SELECT DISTINCT Nombre, Edad **FROM** tabla_clientes**;**

Vuelve a copiar la tabla copy.csv a *tabla_clientes*. ¿Qué pasa si aplicamos la siguiente instrucción?
SELECT DISTINCT $*$ **FROM** tabla_clientes**;**

El comando **WHERE** nos permite seleccionar únicamente aquellas subtablas donde se cumple alguna condición. Es decir, nos permite realizar filtros cruzados.

SELECT ⟨⟨nombre de la columna⟩⟩
FROM ⟨⟨nombre de la tabla⟩⟩
WHERE ⟨⟨condición⟩⟩**;**

Práctica

De la tabla *tabla_clientes*, selecciona los nombres cuyas edades son 25; selecciona los nombres y las edades cuyas edades son mas de 25; selecciona la tabla cuyo primer nombre es Gabriela.

Selección de una columna
SELECT Nombre **FROM** tabla_clientes **WHERE** Edad = 25**;**

Selección de varias columnas
SELECT Nombre, Edad **FROM** tabla_clientes **WHERE** Edad > 25**;**

Selección de todas las columnas
SELECT ∗ **FROM** tabla_clientes **WHERE** Nombre = 'Gabriela'**;**

Los operadores lógicos **OR**, **AND** y **NOT** pueden combinarse con con **WHERE** para realizar filtros con condiciones múltiples. Recordemos que:

Una sentecencia de tipo P_1 ***o*** P_2 ***o ... o*** P_n *es verdadera cuando* ***alguna*** *de sus componentes lo es.*

Una sentecencia de tipo P_1 ***y*** P_2 ***y ... y*** P_n *es verdadera cuando sus* ***todas*** *sus componentes lo son.*

Una sentecencia de tipo ***No*** P *es verdadera cuando* P *es falsa.*

Práctica

De la tabla *tabla_clientes*, muestra aquellos nombres, apellidos y edades de clientes mayores a 20 años pero menores a 30 años; muestra aquellos nombres, apellidos y edades de los clientes con a lo más 25 años o mayores a 30, cuyo primer nombre es Gabriela; muestra aquellos nombres, apellidos y edades de los clientes con a lo más 25 años, o mayores a 30 cuyo primer nombre es Gabriela.

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE** Edad > 20 **AND** Edad < 30**;**

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE** (Edad <= 25 **OR** Edad > 30) **AND** Nombre = 'Gabriela'**;**

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE** Edad <= 25 **OR** (Edad > 30 **AND** Nombre = 'Gabriela')**;**

Práctica

De la tabla *tabla_clientes*, muestra aquellos nombres, apellidos y edades de clientes que no tienen 25 años; muestra aquellos nombres, apellidos y edades de los clientes que no tienen 25 y que no se llaman Jacobo.

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE NOT** Edad = 25**;**

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE NOT** Edad = 25 **AND NOT** Nombre = 'Jacobo'**;**

Recordatorio. Este último también se puede resolver si recordamos las Leyes de De Morgan: (No P) y (No Q) es igual a No (P o Q); (No P) o (No Q) es igual a No (P y Q):

SELECT Nombre, Apellido, Edad **FROM** tabla_clientes **WHERE NOT** (Edad = 25 **OR** Nombre = 'Jacobo')**;**

Supongamos que queremos actualizar los campos de un registro. Para ello, utilizamos **UPDATE** junto con **WHERE**:

UPDATE ⟨⟨nombre de la tabla⟩⟩
SET ⟨⟨nombre de la columna⟩⟩=⟨⟨valor⟩⟩, ⟨⟨nombre de la columna⟩⟩=⟨⟨valor⟩⟩,...
WHERE ⟨⟨condición⟩⟩**;**

Práctica

Actualizar una sola fila especificando una o varias columnas
UPDATE tabla_clientes **SET** Apellido = 'Pe', Edad = 17 **WHERE** Id_cliente = 2**;**

Actualizar múltiples filas
UPDATE tabla_clientes **SET** correo = 'gee@xyz.com' **WHERE** Nombre = 'Gabriela' **OR** Nombre = 'gabriela'**;**

Actualizar todos los valores de una columna
UPDATE tabla_clientes **SET** correo = 'gee@xyz.com'**;**

Para borrar registros de una tabla utilizamos el comando **DELETE** junto con la palabra clave **WHERE** (ESTO ES IMPORTANTÍSIMO).

DELETE FROM ⟨⟨nombre de la tabla⟩⟩
WHERE ⟨⟨condición⟩⟩

Práctica

De la tabla *tabla_clientes* elimina el registro cuyo identificador es 6; elimina aquellos clientes cuyas edades son mayores que 25.

Borrar una sola fila
DELETE FROM tabla_clientes **WHERE** Id_cliente = 6**;**

Borrar varias filas
DELETE FROM tabla_clientes **WHERE** Edad > 25**;**

*Borrar **todas** las filas*
DELETE FROM tabla_clientes**;**

La estructura de una tabla se refiere al número de columnas, los nombres de las columnas, los tipos de columnas y las restricciones de las columnas. Para cambiar esta estructura hacemos uso del comando **ALTER**.

ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
(⟨⟨especificar las acciones⟩⟩)

Las acciones se pueden clasificar de la siguiente manera:

- **Columnas:** añadir **ADD**, borrar **DROP**, modificar **ALTER** o renombrar **RENAME**.
- **Condiciones:** añadir **ADD**, borrar **DROP**.
- **Indexado:** añadir **ADD**, borrar **DROP**.

Añadir una columna:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ADD ⟨⟨nombre de la columna⟩⟩ ⟨⟨Tipo de datos⟩⟩**;**

Eliminar una columna:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
DROP COLUMN ⟨⟨nombre de la columna⟩⟩**;**

Cambiar el tipo de datos de una columna:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ALTER COLUMN ⟨⟨nombre de la columna⟩⟩ **TYPE** ⟨⟨Nuevo tipo de datos⟩⟩**;**

Cambiar el nombre de una columna:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
RENAME COLUMN ⟨⟨nombre de la columna⟩⟩ **TO** ⟨⟨Nuevo nombre de la columna⟩⟩**;**

Práctica

Añadir una columna nueva de tipo caracter variable
ALTER TABLE tabla_clientes **ADD** prueba varchar**;**

Eliminar la columna prueba
ALTER TABLE tabla_clientes **DROP COLUMN** prueba**;**

Cambiar el tipo de dato de la columna Edad
ALTER TABLE tabla_clientes **ALTER COLUMN** Edad **TYPE** varchar**;**

Cambiar el nombre de la columna correo
ALTER TABLE tabla_clientes **RENAME COLUMN** correo **TO** correo_cliente**;**

Agregar o cambiar una condición:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ALTER COLUMN ⟨⟨nombre de la columna⟩⟩ **SET NOT NULL;**

Eliminar una condición:
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ALTER COLUMN ⟨⟨nombre de la columna⟩⟩**DROP NOT NULL;**

Agregar una condición que deben cumplir los datos
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ADD CONSTRAINT ⟨⟨nombre de la columna⟩⟩ **CHECK** ⟨⟨nombre de la columna⟩⟩ >=100**;**

Convertir una columna a llave primaria
ALTER TABLE ⟨⟨nombre de la tabla⟩⟩
ADD PRIMARY KEY ⟨⟨nombre de la columna⟩⟩**;**

Práctica

De la tabla *tabla_clientes*, pedir que la columna de identificadores no admita valores nulos y luego quitar esa condición; pedir que la columna de Id_cliente admita solo valores mayores a 0; hacer que la columna de identificadores sea llave primaria.

ALTER TABLE tabla_clientes **ALTER COLUMN** Id_cliente **SET NOT NULL;**
INSERT INTO tabla_clientes(Nombre, Apellido, Edad, correo) **VALUES** ('aa','bb',25,'ab@xyz.com') **;**

ALTER TABLE tabla_clientes **ALTER COLUMN** prueba **DROP NOT NULL;**

ALTER TABLE tabla_clientes **ADD CONSTRAINT** Id_cliente **CHECK** (Id_cliente>0)**;**
INSERT INTO tabla_clientes **VALUES** (-1,'cc','dd',67,'ab@xyz.com') **;**

ALTER TABLE tabla_clientes **ADD PRIMARY KEY** Id_cliente **;**