**Complementario: PostgreSQL**

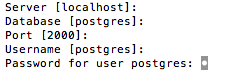
**Interacción con PostgreSQL desde la Consola**

Existen dos maneras en las que podemos acceder a todas nuestras bases de datos en PostgreSQL: Consola (terminal en Mac) e Interfaz gráfica, también conocida como “Manejador de base de datos”. Hay diferentes *Manejadores de base de datos* para PostgreSQL, pero el más usado es **pgAdmin.** Si bien las interfaces gráficas llegaron para facilitarnos todo, el uso de la consola siempre será primordial; pues, muchas veces vas a encontrar que la Interfaz gráfica presenta problemas (o *bugs*) y son problemas que sólo se podrán solucionar con la consola. Dicho lo anterior, vamos a aprender a cómo acceder a nuestras bases de datos desde la consola y cómo usar los comandos más útiles.

**Acceso a tus bases de datos desde consola:**

1. Abre SQL Shell (psql), viene con la instalación de PostgreSQL. Es de esta manera que se accede a la consola de de PostgreSQL, desde SQL Shell.

Cuando abra SQL Shell, verá algo así:



**Expliquemos uno a uno.**

En **Server [localhost]:** Si tienes algún servicio diferente instalado en otro equipo, aquí debes escribir la *IP* o debes escribir el nombre del *dominio* donde está alojado tu servidor. Si no hay ninguna modificación en esa configuración simplemente presionas *Enter,* ya que al presionarlo el servicio toma a *localhost* por defecto, es decir, *localhost* es nuestro valor predeterminado (el de nuestra computadora).

En **Database [postgres]:** Se te pide el nombre de la base de datos, la base de datos a la que te gustaría acceder; es decir, ya debe estar creada previamente por ti (por ti si estás desde *localhost.* Si estás situado sobre otra “computadora”, porque pasaste la *IP* o el *dominio* de otro Server, pasas el nombre de la base de datos que está ubicada en esa otra computadora). En caso que no haya creado ninguna base de datos, ni pertenezca a otro Server, presione *Enter* sin más; PostgreSQL, por defecto, accederá a su propia base de datos predeterminada, ésta recibe el nombre de: ***postgres*** (viene con la instalación).

En **Port [2000]:** Se escribe el puerto, especialmente en los casos en los que deseas moverte sobre un puerto diferente al puerto predeterminado (nuestro puerto por defecto es 2000). Si deseas conservar el puerto predeterminado simplemente presiona *Enter* sin más.

En **Username** **[postgres]:** Se escribe el nombre del usuario. Sin embargo éste, al igual que la base de datos inicial, fue creado de forma predeterminada. El nombre de usuario predeterminado recibe el mismo nombre de la base de datos; es decir, ***postgres*.** No los confunda, si bien tienen el mismo nombre, no son lo mismo. Si no ha instalado ni configurado ningún otro usuario, y presiona *Enter* sin más, PostgreSQL tomará el nombre de usuario por defecto; es decir, ***postgres.***

En **Password for user postgres:** Se pasa la clave que le debió haber asignado, en un principio, a su usuario (en este caso *postgres*). **Es recomendable siempre asignar una clave a su usuario porque de no hacerlo representaría un alto riesgo de seguridad. Inserte clave y presione *Enter.***

**Sobre cómo usar los comandos más útiles.**

Posteriormente, al introducir su clave y presionar *Enter*, verá el nombre de la base de datos en la que está posicionado actualmente y, a su lado, el cursor listo para digitar sus comandos. Tal que así:



**Comandos:**

1. ENTRA A LA CONSOLA DE POSTGRES:  
   **psql -U postgres -W**
2. MUESTRA LOS COMANDOS \ DE POSTGRES:  
   **\?**
3. LISTA TODAS LAS BASES DE DATOS:  
   **\l**
4. MUESTRA LAS TABLAS DE UNA BASE DE DATOS:  
   **\dt**
5. CAMBIAR A OTRA DB:   
   **\c nombre\_DB**
6. DESCRIBE LA ESTRUCTURA DE UNA TABLA:  
   **\d nombre\_tabla**
7. MUESTRA TODOS LOS COMANDOS DE SQL PROPIAMENTE:  
   **\h**
8. MUESTRA COMO OPERA UN COMANDO DE SQL (TENER UNA DESCRIPCIÓN):  
   **\h nombre\_de\_la\_funcion**
9. CANCELA TODO *OUTPUT* EN EJECUCIÓN (DETIENE PROCESOS PARA LLEVARTE DE NUEVO AL PUNTO DE PARTIDA, DE LA TERMINAL, Y CORRER NUEVO CÓDIGO):  
   **Ctrl + C**
10. MUESTRA LA VERSIÓN DE POSTGRES INSTALADA, DEBE PONER EL ';' AL FINAL:  
    **SELECT version();**
11. VUELVE A CORRER LA ÚLTIMA FUNCIÓN EJECUTADA:  
    **\g**
12. ACTIVA EL CONTADOR DE TIEMPO PARA QUE LA CONSOLA TE DIGA EN CADA EJECUCIÓN CUÁNTO DEMORÓ EN EJECUTAR DICHA FUNCIÓN:  
    **\timing**
13. LIMPIA LA PANTALLA DE LA CONSOLA PSQL:  
    **Ctrl + L**

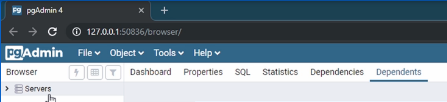
Tenga en cuenta que si desea crear una nueva base de datos desde su consola para luego usarla en su interfaz gráfica debe primero, después de crear la base de datos en consola (**CREATE DATABASE nombre\_database;**), desplazarse a ella tal que así: **\c nombre\_database;**

Al situarse o desplazarse desde su consola a la base de datos de su interés, ésta ya estaría habilitada dentro de su manejador de datos (pgAdmin).

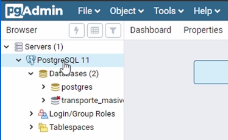
**Algo muy importante:** La base de datos de *PostgreSQL* funciona con SQL estándar.

**Interacción con PostgreSQL desde la Interfaz gráfica (pgAdmin)**

Para acceder a nuestras bases de datos desde la interfaz gráfica pgAdmin debes simplemente abrir la aplicación, tal que así.

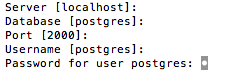


A tu izquierda podrás ver todos los servicios con respecto a tus conexiones de bases de datos (***Servers***); es decir, podrás ver todas las bases de datos a las que tienes conexión. Desde ***servers*** también se podrán alistar tanto las bases de datos como sus tablas correspondientes; más precisamente, desde su ítem *PostgreSQL* (este es el *Server* por defecto, es desde aquí que podrá ver –en un principio- todas sus bases de datos creadas como las bases de datos predeterminadas, incluido usuarios). **Veamos**:

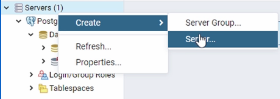


La base de datos “*transporte\_masivo*” fue creada por nosotros y la base de datos “*postgres*” vino por defecto. De estas bases de datos también se pueden desprender sus respectivas funciones y tablas (*schemas*).

Antes de continuar, **qué es o qué hace en práctica un *Server?***Un***Server*** prácticamente es una instancia desde la cual usted puede abrir todas sus bases de datos y manipularlas de una forma particular, una forma particular para cada *Server*, por ejemplo. Lo dicho, generalmente la intención de crear varios *Servers* es poder tener la posibilidad de poder tener varios controles de versión sobre sus mismas bases de datos; es decir, que cada *server* represente y tenga su propia edición de las mismas bases de datos que usted posee. Son como distintas copias de seguridad que se crean, sobre las mismas bases de datos, para tener su propia edición de bases de datos en cada una de ellas. *Entonces…*

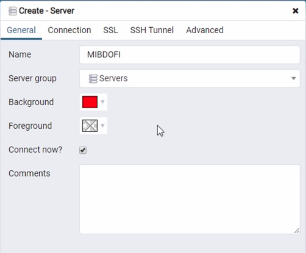
Recuerde esto: 

Si quisiéramos hacer lo anterior, crear una *conexión* o *acceso (****server****),* tal como lo hicimos desde consola pero ahora desde la interfaz gráfica ***pgAdmin,*** siga los siguientes pasos:



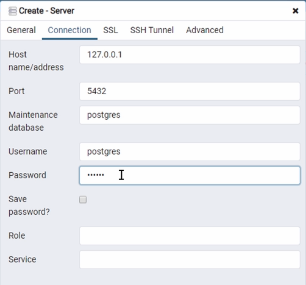
Hacemos clic en la opción “***Server***…”

Pestaña **General**



***Name*** es el equivalente al nombre que recibirá su ***Server***. En ***Background***se le puede asignar un color al *Server* que se está creando; es decir, como para tener una forma más visual de identificar cada *Server* en sí.

Pestaña **Connection**



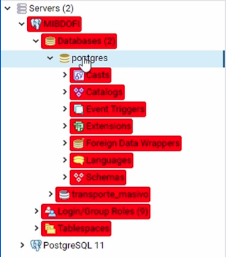
***Host name/address*** es el equivalente a ***Server[localhost]*** en la consola, ***Port*** equivalente a ***Port[number]***, ***Maintenance database*** equivalente a ***Database[postgres]***, ***Username*** equivale a ***Username[postgres]* & *Password*** equivale a **Password for user postgres.**

Si presiona guarda se crea inmediatamente otra conexión o ***Server.***

**Funciones & entidades de una base de datos**

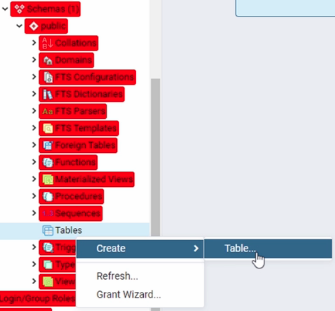
Recuerda esto *“De estas bases de datos también se pueden desprender sus respectivas funciones y tablas (schemas)”…*

Pues bien, nos referimos visualmente a esto:



En el anterior ejemplo podrá observar que, de la base de datos ***postgres***, estamos desprendiendo varios *ítems;* sí, se trata de todas las funciones que posee una base de datos. Vamos una por una.

* **Casts**: Sirve para hacer traducción de tipos de datos explícitos.
* **Catalogs**: Sirve para catalogar la información, es una especie de subdirectorio de tablas.
* **Event Triggers:** Sirve para ejecutar funciones a partir de acciones que se presencian en la base de datos.
* **Extensions**: Nos servirá para instalar módulos desarrollados por terceros para nuestra base de datos.
* **Foreign Data Wrappers**: Acceso a base de datos remoto.
* **Languages**: Hace referencia a los lenguajes de programación que puedes usar dentro de la base de datos.
* **Schemas**: Nos permite ver la información propia a las tablas de nuestra base de datos.

***Caso****: creación de tablas desde la interfaz grafica de Postgres.* Sitúese en su DB de interés y vaya a *Schemas.* 

*Más adelante explicaremos más a fondo estos conceptos para darles más contexto y tener una mayor apropiación del tema.*

Adicionalmente, **pgAdmin** nos permite ver en tiempo real todo lo que sucede con nuestra base de datos. Si hacemos clic en “Databases”; nos mostrará a la derecha, en una barra de menú, distintas pestañas:

* **Dashboard**: Nos muestra las conexiones simultaneas, las transacciones, lo que está escribiendo, lo que está leyendo, en fin, lo que sucede en tiempo real. En la parte inferior, en “Server Activity”, podemos ver las funciones que se están ejecutando para la base de datos que estamos visualizando.
* **Properties**: Nos muestra las propiedades de nuestra DB, como los tipos de datos, el propietario de la base de datos, comentarios, entre otros.
* **Statistics**: Nos muestra datos estadísticos de nuestra DB.
* **Dependencies**: Nos muestra las dependencias que tenga, en dado caso, nuestra base de datos.

**Para aclarar:** Igualmente podemos ejecutar consultas SQL en pgAdmin, nos posicionamos en la base de datos donde queramos realizarlas (por ejemplo, *postgres*) y seleccionamos la opción *Tools* del menú superior*,* posteriormente se hace clic en ***Query Tool****.*

***Query Tool***es el equivalente al asistente gráfico, de cualquier tipo de manejador de base de datos, donde se escriben y se corren todas nuestras *Queries,* es decir, es el editor de consultas SQL. Su estándar es el mismo: Las palabras claves propias del lenguaje SQL se deben escribir en mayúsculas y el final de cada *Query* se debe terminar con un punto y coma (;).

**Extra - Archivos de configuración**

A través de la sentencia **SHOW config\_file;** se nos muestra donde están los archivos de configuración.

En mi caso la ruta es: **"/Library/PostgreSQL/9.5/data/postgresql.conf"** *(Varia según la versión de su pgAdmin)*

Algo a tener en cuenta es que no se puede acceder a la ruta de instalación por *default* debido a falta de permisos. Para ingresar basta con correr un: **sudo cd /Library/PostgreSQL/9.5/data/**

**Postgresql.conf**: Configuración general de postgres, múltiples opciones referentes a direcciones de conexión de entrada, memoria, cantidad de hilos de procesamiento, replica, etc.

**pg\_hba.conf**: Muestra los roles así como los tipos de acceso a la base de datos.

**pg\_ident.conf**: Permite realizar el mapeo de usuarios. Permite definir roles a usuarios del sistema operativo donde se ejecuta postgres.

**Más y más comandos para psql, terminal interactiva para trabajar con PostgreSQL**

**Comandos de ayuda**

Los dos principales comandos que nos permiten revisar, y pedir ayuda, a los demás comandos y consultas son: **\?** & **\h**

**\?**,con el cual podemos ver la lista de todos los comandos disponibles en consola, sólo alista los comandos que empiezan con backslash ( \ ).

**\h**, con este comando veremos la información de todas las consultas SQL disponibles en consola, sirve también para buscar ayuda sobre una consulta específica. Para buscar información sobre una consulta específica basta con escribir **\h** seguido del inicio de la consulta de la que se requiera ayuda, así: **\h ALTER**

De esta forma podemos ver toda la ayuda disponible con respecto a la consulta ***ALTER***

**Comandos de navegación y consulta de información**

**\c <data\_base\_name>** Moverse a una base de datos en especifica

**\l**Listar base de datos disponibles

**\dt**Listar las tablas de la base de datos actual

**\d <table\_name>**Describir una tabla

**\dn**Listar los esquemas (tablas, entidades) de la base de datos actual

**\df**Listar las funciones disponibles de la base de datos actual

**\dv**Listar las vistas de la base de datos actual

**\du** Listar los usuarios y sus roles de la base de datos actual

**Comandos de inspección y ejecución**

**\g** Volver a ejecutar el comando ejecutado justo antes

**\s** Ver el historial de comandos ejecutados

**\s <file\_name>** Si se quiere guardar la lista de comandos ejecutados en un archivo de texto plano

**\i <file\_name>** Ejecutar los comandos desde un archivo

**\e** Permite abrir un editor de texto plano, escribir comandos y ejecutar en lote. En detalle, **\e** le permite abrir un editor de texto plano para que usted escriba allí todos sus comandos; luego, cuando finalice, guardará los cambios y cierra,… al cerrar el editor se ejecutarán todos los comandos guardados.

**\ef** Equivalente al comando anterior pero permite editar también funciones en PostgreSQL.

**Comandos para Debug y optimización**

**\timing** Activar / Desactivar el contador de tiempo por consulta.

**Comandos para cerrar la consola**

**\q** Cerrar la consola

**Introducción a Tipos de datos**

Este es un curso complementario y ya debe de tener conocimiento de los principales *tipos de datos* que se manejan en una base de datos relacional, ahora simplemente los vamos a mencionar: **Numéricos, Texto, Fecha y Hora & Boolean,** también se habla de tipos de datos **Binarios (1 & 0) & Monetarios.** Sin embargo, hay otros *formatos* (tipos de datos) que sólo se manejan dentro del entorno de trabajo propio a PostgreSQL, estos *nuevos* tipos de datos son llamados **Especiales,** éstos son: **Geométricos** (te permiten, al usar *ejes X e Y*, calcular distancias y áreas)**, Dirección de Red** (te permiten almacenar *IPs* y hacer cálculos de mascaras de red)**, Texto tipo Bit** (te permiten hacer cálculos en otros sistemas, por ejemplo, hexadecimal o binario), **XML y JSON** (las *APIs* se comunican por medio de estos dos lenguajes y son formatos aceptados por Postgres) **& Arreglos** (Se refiere a la formación de Vectores y Matrices, también son aceptados en Postgres).

**Jerarquía de bases de datos**

**Toda jerarquía de base de datos se basa en los siguientes elementos:**

* **Servidor de base de datos:**Computador que tiene un motor de base de datos instalado y en ejecución.
* **Motor de base de datos:**Software que provee un conjunto de servicios encargados de administrar una base de datos.
* **Base de datos:**Grupo de datos que pertenecen a un mismo contexto.
* **Esquemas de base de datos en PostgreSQL:**Grupo de objetos de base de datos que guarda relación entre sí (tablas, funciones, relaciones, secuencias).
* **Tablas de base de datos:**Estructura que organiza los datos en filas y columnas formando una matriz.

**Teoría en práctica: Crearemos de una base de datos relacional**

Se pretenderá crear la estructura de una base de datos relacional que modele un sistema de transporte masivo (tomando como ejemplo el transporte de *trenes* y todas las demás entidades que interactúan en dicho sistema).

La base de datos se llamará ***transporte***, usaremos su esquema predeterminado *public*.

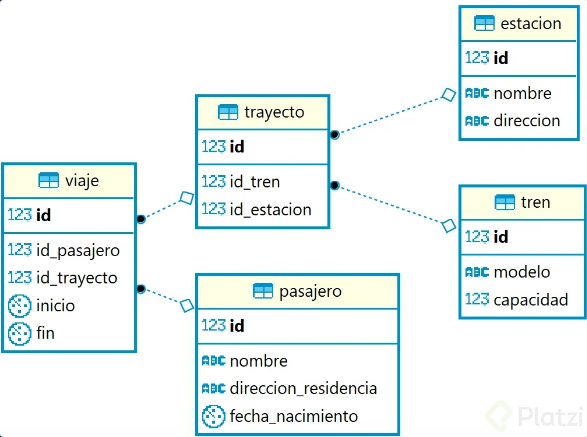
**El esquema *public* contiene las siguientes tablas:**

* Pasajeros
* Estaciones
* Trenes

**Y las tablas de relaciones entre cada uno de los elementos anteriores son:**

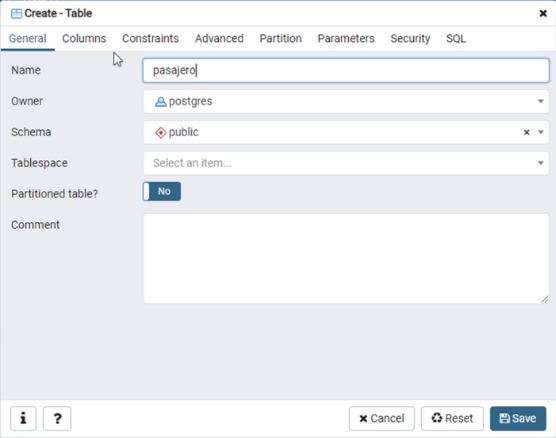
* Trayectos
* Viajes

**El esquema relacional entre las tablas corresponde al siguiente diagrama:**

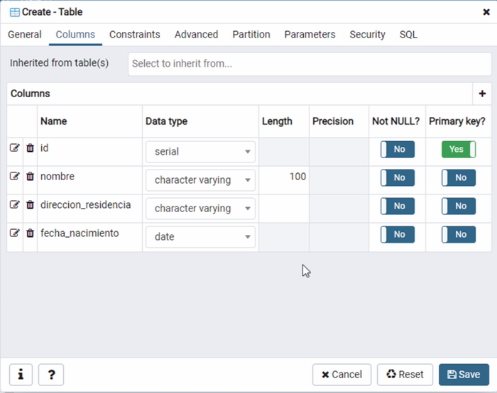


**Al grano…**

1ra Tabla: ***Pasajeros***

****

Creación de ***Columnas*** para la 1ra tabla ***Pasajeros***



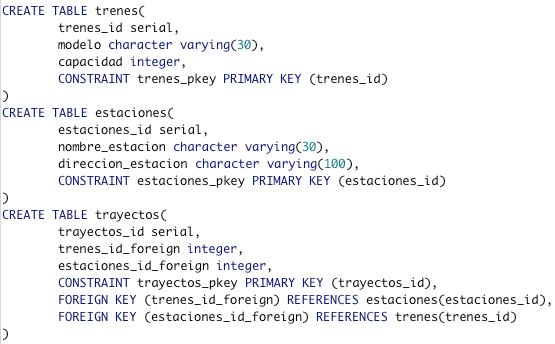
*1ra Columna:* ***pasajeros***\_***id*** (se define con un tipo de dato ***serial*** y conservará la ***Primary key***).

El tipo de datos ***serial,*** propio de postgres, es el equivalente a un **Integer** con una función ***Auto Increment*** dentro de MySQL. Ahora, por esa misma razón, la columna o atributo con este tipo de dato asignado, *serial*, muy generalmente siempre será la que será definida también como ***Primary Key***.

***Aclaración***: esta primera columna fue definida como “**id**”, mas idealmente debe ser: ***pasajeros***\_***id.***

***Tenga en cuenta algo:*** Si alguna de sus *primary keys*, definidas con un tipo de dato *serial*, serían luego *exportadas* y usadas como *foreign keys* dentro de otra tabla; debe saber que, en esa otra tabla, el tipo de dato **no se define como un tipo de dato *serial***; si no, como ***integer****…* No nos interesa que un atributo o campo que se está definiendo como *foreign key* tenga la función de auto-incremento sobre su tipo de dato *entero* por cada registro que se haga en él, no es correcto.

Observe, por ejemplo, cómo se definen (con qué tipo de dato) las *foreign keys* dentro de las tablas que importan las *primary keys* de otras (otras tablas, me refiero):



Ya usted debe saber de paso que, por *buenas prácticas*,se evita repetir nombres de campos o atributos en otras tablas,… lo ideal es que siempre deje claro una distinción.

*2da Columna:* ***nombre*** (se define con un tipo de dato ***character varying***).

El tipo de datos ***character varying***, propio de postgres, es exactamente el mismo a ***varchar(n)*** en MySQL.

Debajo de la casilla ***Length*** se asigna la cantidad de números o caracteres, depende del tipo de dato, que puede recibir el atributo o columna en cuestión; esto, por cada registro (es opcional limitar la longitud de la cantidad numérica de valores que puede admitir un atributo; es decir, **puede dejar esta casilla en blanco**… sólo se define si tiene la intención anticipada de limitar la cantidad numérica de valores que pueden ser pasados al atributo, por cada registro del mismo, y pretende optimizar memoria dinámica con ello). Para que me entienda, por ejemplo, ***Length*** definiría el ***“n”*** de un tipo de dato ***varchar(n)***; es decir, en ese caso, definiría la cantidad de caracteres que podría tener, como máximo, la cadena de texto ***varchar(n).***

Ahora, como no estamos hablando en términos de ***varchar(n)***,sino, ***character varying***;en el caso puntual de la columna ***nombre***, la cantidad máxima de caracteres que podría recibir dicho atributo (campo o columna), por cada uno de sus registros, sería de: ***100.***

*3ra Columna:* ***direccion\_residencia*** (se define con un tipo de dato ***character varying***).

*4ta Columna:* ***fecha\_nacimiento*** (se define con un tipo de dato ***date****… ya conoce este tipo de dato*).

Definición de ***Constraints*** (restricciones) para las columnas de la 1ra tabla ***Pasajeros***

En primer lugar, sólo vamos a dejar señalado un ***Constraint***y éste será el de la *Primary Key.* Por la anterior razón es que, además de estar sobre la pestaña ***Constraints***, estamos por debajo de la sub-pestaña ***Primary Key.***

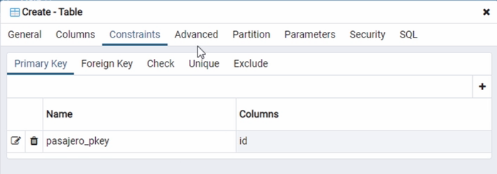
******

Dicho eso, procedamos:

No solo basta con haber definido la ***Primary key***, desde la pestaña de ***Columns***, para una columna en cuestión; no, sino que también debe llamarse a la columna determinada en la pestaña ***Constraints*** y dentro de la sub-pestaña ***Primary Key,*** más precisamente debajo de la casilla ***Columns*** (sería ***ID*** en la imagen)*.*

Ahora, debajo de la casilla ***Name*** (dentro de la misma pestaña ***Constraints*** y dentro de la sub-pestaña ***Primary Key***)se debe pasar el nombre de la tabla abordada en cuestión; en este caso, ***pasajeros*** + los caracteres claves: \_***pkey***; quedando algo así:***pasajeros\_pkey***.

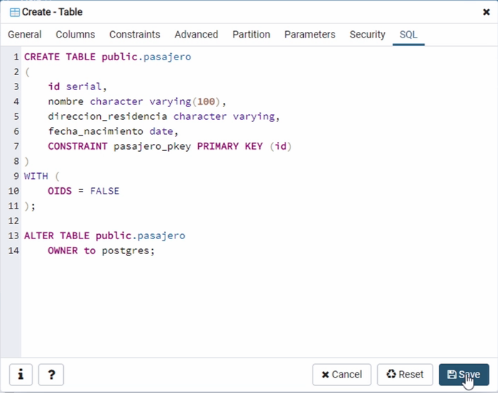
Veamos:



**Aclaración:** Si bien sólo definimos un solo ***Constraints*** desde su misma pestaña; en este caso, sólo se definió la restricción relacionada a la ***Primary Key***, no siempre es así***…*** Lo ideal es que para todos los casos, todas las columnas que lo requieran porque tengan restricciones, siempre se necesite asignar las restricciones desde la pestaña ***Constraints***, incluso aunque haya hecho definiciones de éstas desde la pestaña ***Columns***.

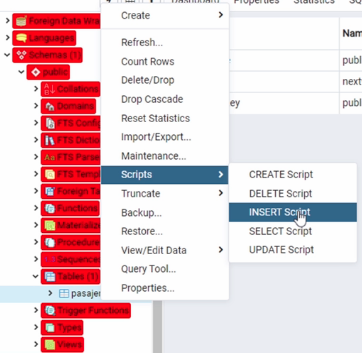
**Nota**: En la captura se ve **“pasajero\_pkey”** sin la “s” al final de *pasajeros;* sin embargo, como ya debe saber, lo ideal es definir siempre sus entidades en plural (en nuestro caso al tutor se le olvidó, pero nosotros lo haremos con plurales desde nuestra interfaz gráfica), tal que así: **“pasajeros\_pkey”.**

Ahora, en caso que ya no pretenda asignar una configuración adicional sobre la creación de su tabla, usted podrá ver el código tras bambalinas desde la pestaña **SQL;** es decir, ver el código crudo de SQL (con sus comandos) que se ejecutará para crear la tabla en cuestión. Este código, sea cual sea su *Query,* siempre podrá visualizarlo si se encuentra ejecutando comandos con la asistencia directa de su interfaz gráfica y no directamente desde su editor o terminal de comandos. Observe:

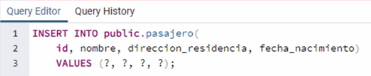


***Save*** y listo! Ya queda nuestra primera tabla ***pasajeros*** creada y estructurada… *Lista para recibir (insertar) valores.*

***Insert*** sobre nuestra 1ra tabla ***pasajeros***



Como nos estamos asistiendo directamente de la capacidad de la interfaz gráfica de pgAdmin, ésta por defecto nos facilita todo… al punto de escribirnos el *Query* prefabricado para el comando de tipo ***Insert***, tal que así:



Esta sintaxis ya la debe entender y, adicionalmente, también debe saber que podría omitir el atributo ***id*** pasado como su posible valor asignado en ***VALUES***, pues, a este atributo se le fue asignado la función ***Auto Increment.*** Debe quedar algo así al reemplazar cada una de las interrogaciones por sus respectivos valores (es una guía, no es la inserción de valores reales):

