# **¿Qué es Postgresql?**

Es un motor de base de datos.

Tenemos que tener en cuenta 3 cosas importantes:

* Lenguaje.
* Motor.
* Servidor.

## **¿Por qué PostgreSQL?**

* Tipos de datos.
* Integridad de datos.
* Concurrencia, rendimiento.
* Fiabilidad, recuperación ante desastres.
* Seguridad.
* Extensibilidad.
* Internalización, búsqueda de texto.
* Sistema de geolocalización integrado (**PostGIS**).

Cumple ACID

* **A** -Atomicity **🡪** atomicidad: Hace referencia al **paralelismo** y **división** que tienen las funciones de forma que podemos ejecutar poco a poco una función de forma paralela y **en caso de que alguna falle**, se hace un **rollback** (se vuelven atrás los cambios), entonces vamos a poder restaurar los datos de vuelta sin dañar nada.
* **C** -Consistency **🡪** consistencia: Todo lo que se desarrolla en la base de datos, en la que los datos tienen **integridad** y **congruencia** entre sí, por lo que cada dato cumple las normas.
* **I** -Atomicity **🡪** (isolation) aislamiento: Aislamiento de cada tarea corriendo en **paralelo**, tengan o no algo en común. Incluyendo incluso la entrada de diferentes usuarios al mismo tiempo.
* **D** -Durability 🡪 durabilidad: Asegura la restauración siempre adecuada, debido a la seguridad aplicada a la base de datos. Por lo que cualquier evento catastrófico no será excusa para perder los datos. Por ejemplo: Supongamos que el servidor se apagó inesperadamente durante una inserción, esa información no se va a perder, porque postgres guarda la información en una bitácora y luego hace el cambio efectivo en la base de datos, sí en ese momento falla el server, la bitácora la lee para recuperar los datos que no pudo insertar.

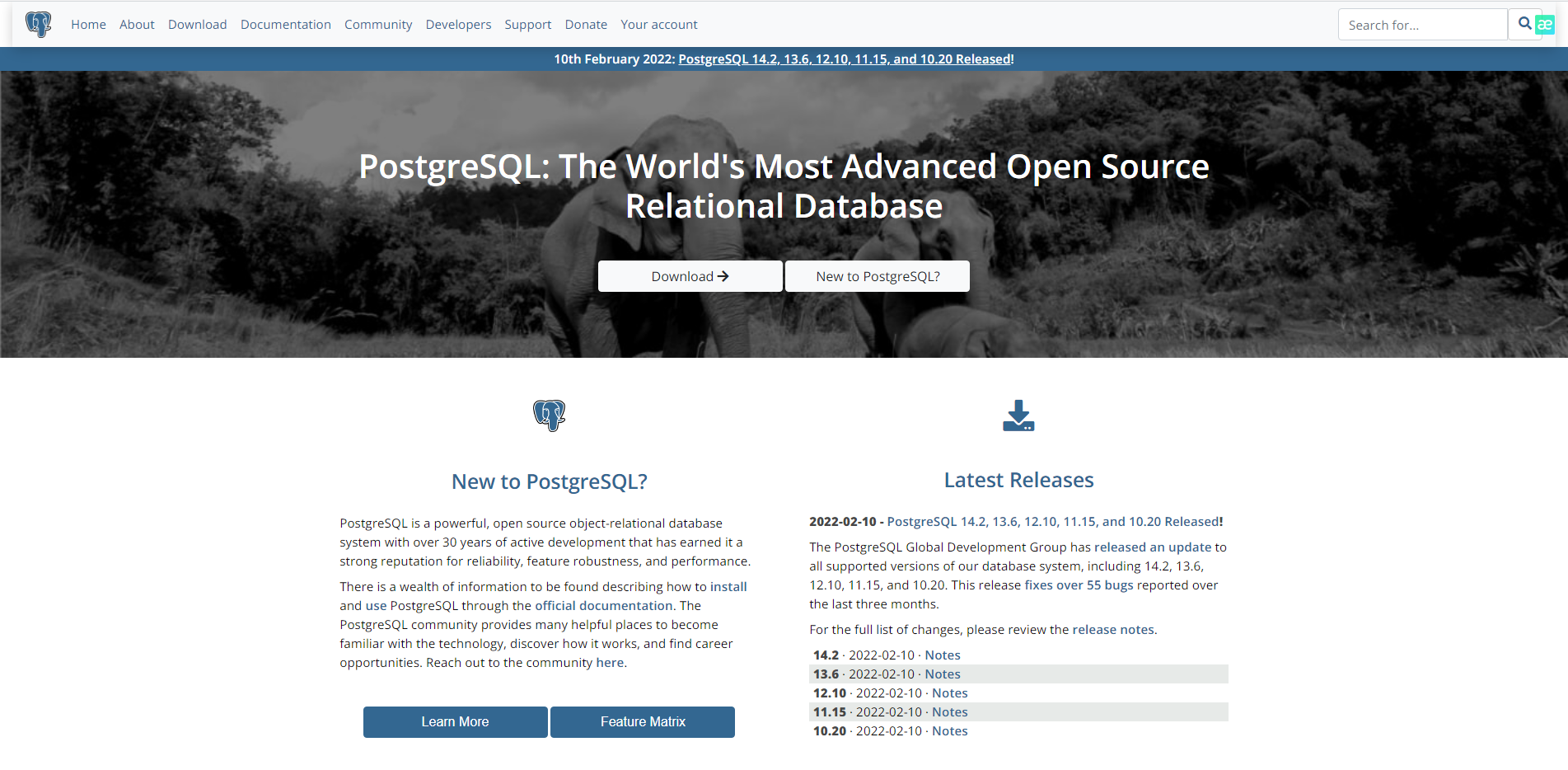
Sitio oficial de PostgreSQL [aquí](https://www.postgresql.org/).

# **Instalación y configuración de la Base de Datos**

Vamos a instalar PostgreSQL en nuestra computadora. A continuación veremos el paso a paso y algunos consejos útiles para instalar y configurar correctamente PostgreSQL en nuestro equipo. En éste caso, usaremos Windows, pero los pasos son bastante similares entre los diferentes sistemas operativos.

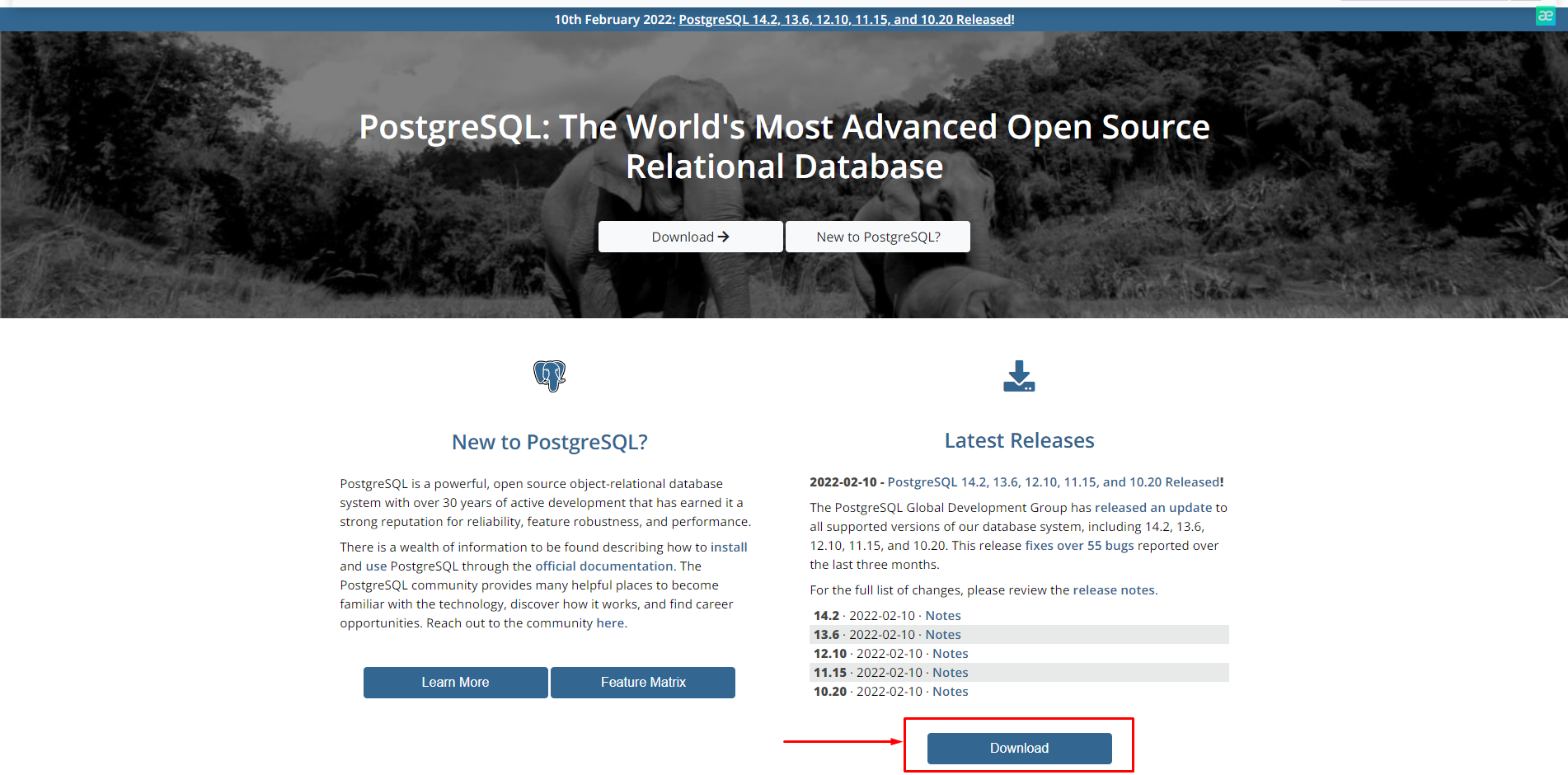
1. Primer paso: ir al [sitio oficial](https://www.postgresql.org/).

Actualmente, la página web oficial de postgres luce así:

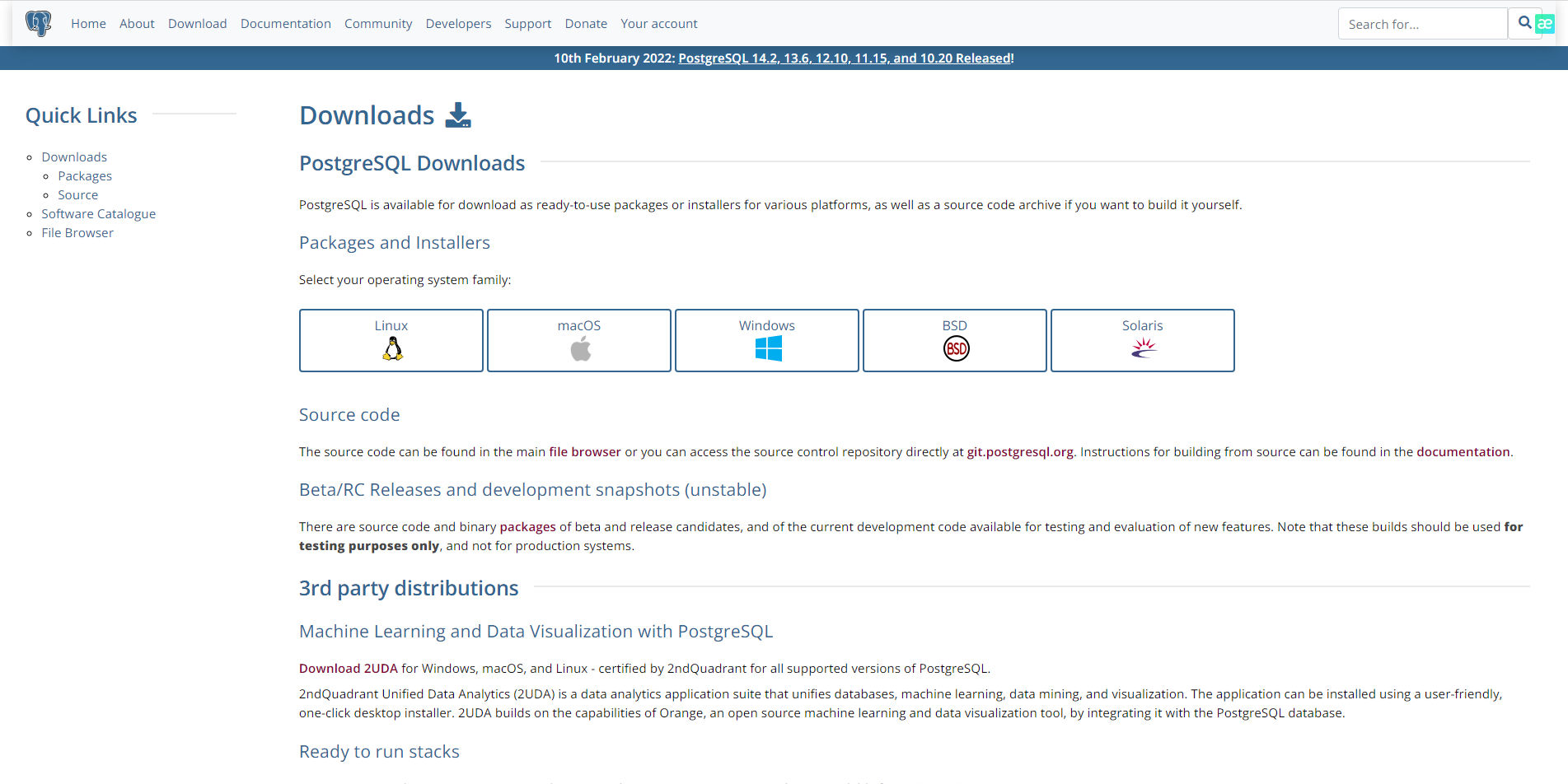


Ten en cuenta que puedes ver esta página en diferentes idiomas, depende de la configuración predeterminada de idioma de tu navegador.

Hacer clic en el botón ‘Download’ (Descarga) que se encuentra en la parte **inferior derecha**.

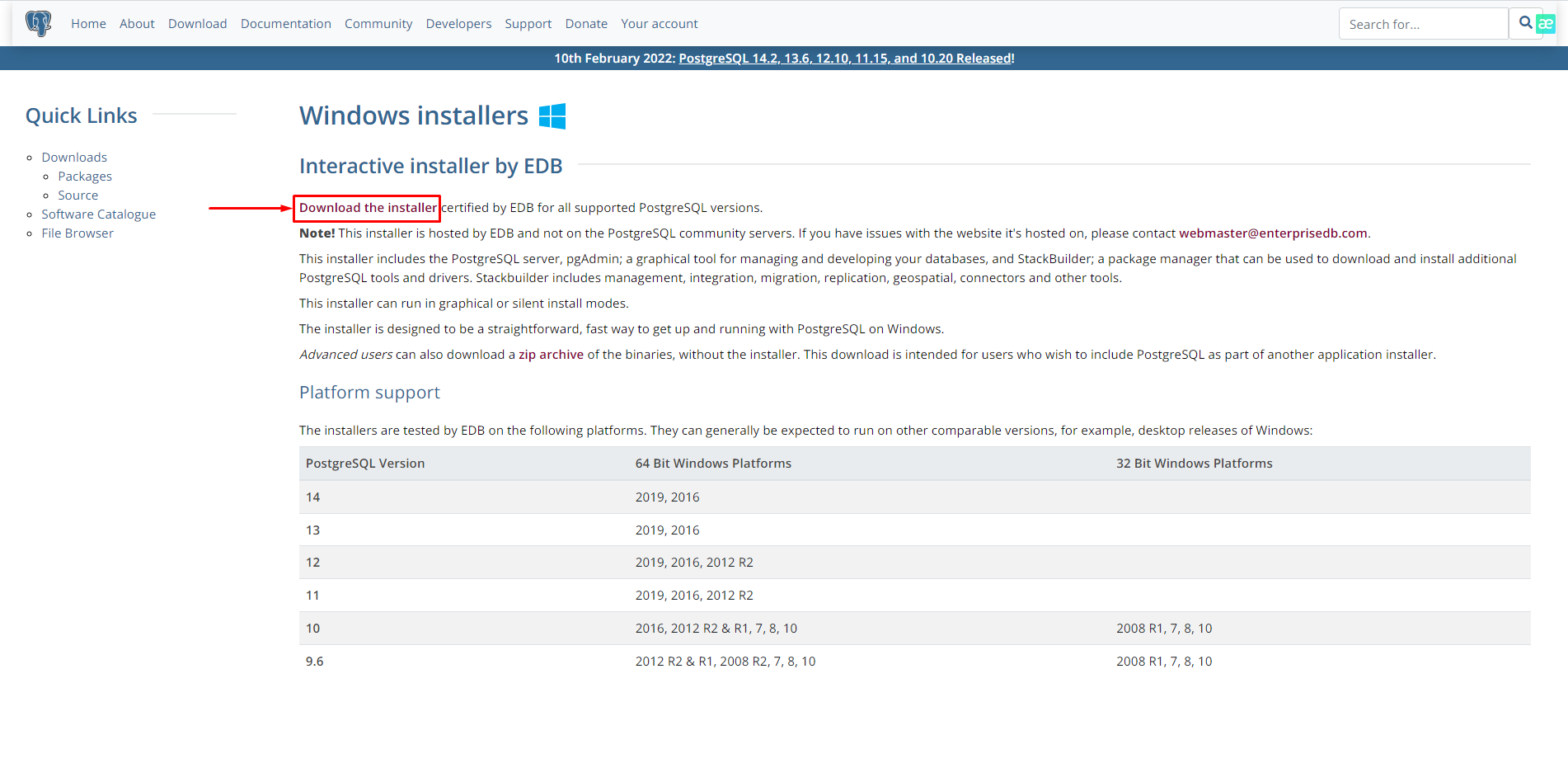


Veremos lo siguiente:

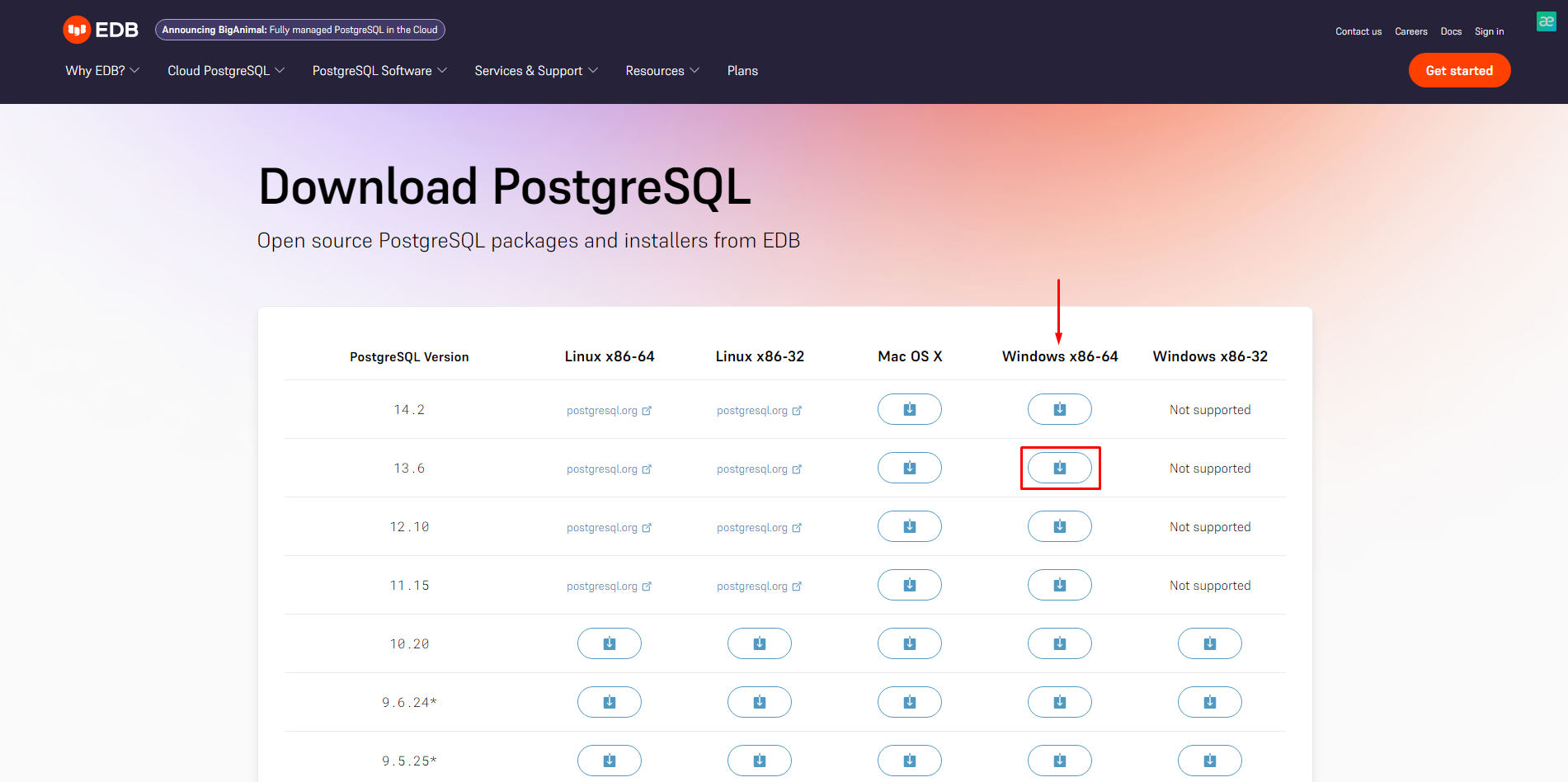


Seleccionamos la opción que corresponda con tu sistema operativo, para este caso hacemos clic en “**Windows**”:

Hacemos click en **Download the installer**.



Esto nos va a llevar a la Web de **Enterprise DB** o EDB. EDB es una empresa que ofrece servicios sobre el motor de base de datos PostgreSQL y ofrece un instalador para Postgres de manera gratuita.



**Es altamente recomendable seleccionar la penúltima o antepenúltima versión**. Si bien la última versión estable está disponible, en este caso la 14.2, no es recomendable instalarla en nuestro equipo, ya que al momento de instalarla o usar un servicio en la Nube para Postgres, lo más seguro es que no esté disponible y sólo esté hasta la versión 13.6, que no es la última versión. Esto es porque todos los proveedores de Infraestructura no disponen de la versión de Postgres más actual siempre (tardan un poco en apropiar los nuevos lanzamientos).

Si tienes un equipo con **Linux**, la instalación la puedes hacer directamente desde los repositorios de Linux, EDB ya no ofrece soporte para instaladores en Linux debido a que se ha vuelto innecesario, el repositorio de Linux con PostgreSQL ofrece una manera mucho más sencilla y estándar para instalar PostgreSQL en linux.

1. Descargamos la versión “**Windows x86-64**” (porque nuestro sistema operativo es de 64 bits). En caso de que tu equipo sea de 32 bits debes seleccionar la opción “**Windows x86-32**”.

Vamos a descargar la versión 13.6. Hacemos clic en Download y guardamos el archivo que tendrá un nombre similar a:

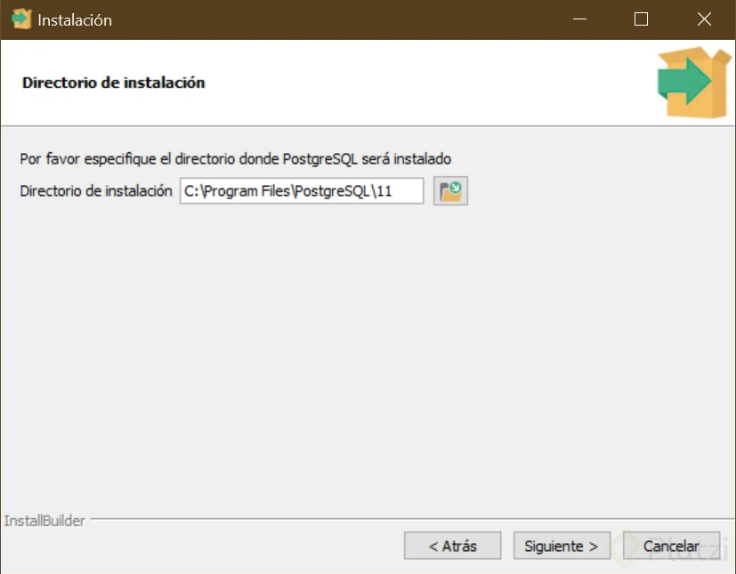
“**postgresql-13.6-2-windows-x64.exe**”

Ahora vamos a la carpeta donde descargamos el archivo **.exe**, debe ser de aproximadamente 190 MB, lo ejecutamos.

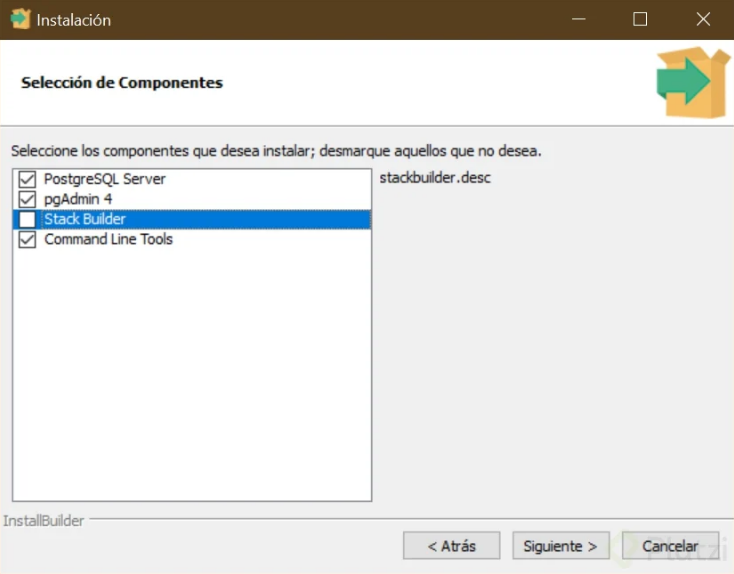
Veremos algo similar como lo siguiente:



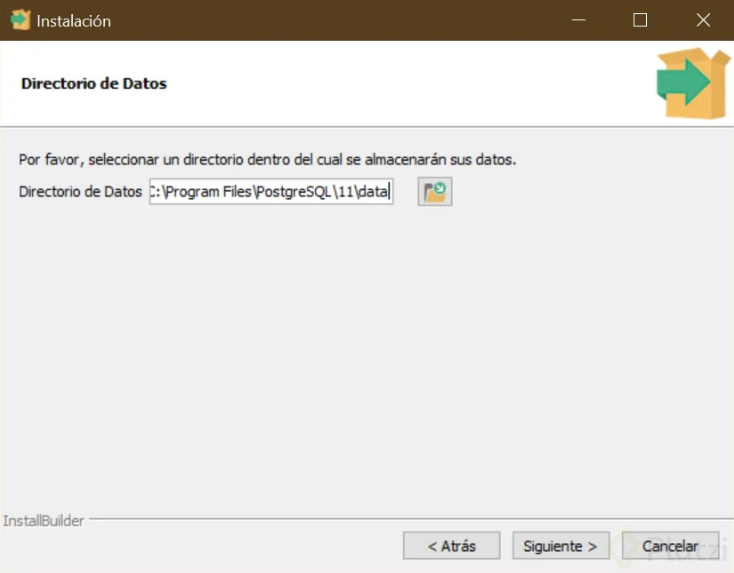
Hacemos clic en siguiente. Si deseas cambiar la carpeta de destino, ahora es el momento:



Seleccionamos los servicios que queremos instalar. En este caso dejamos seleccionados todos menos “**Stack Builder**”, pues ofrece la instalación de servicios adicionales que no necesitamos hasta ahora. Luego hacemos clic en siguiente:

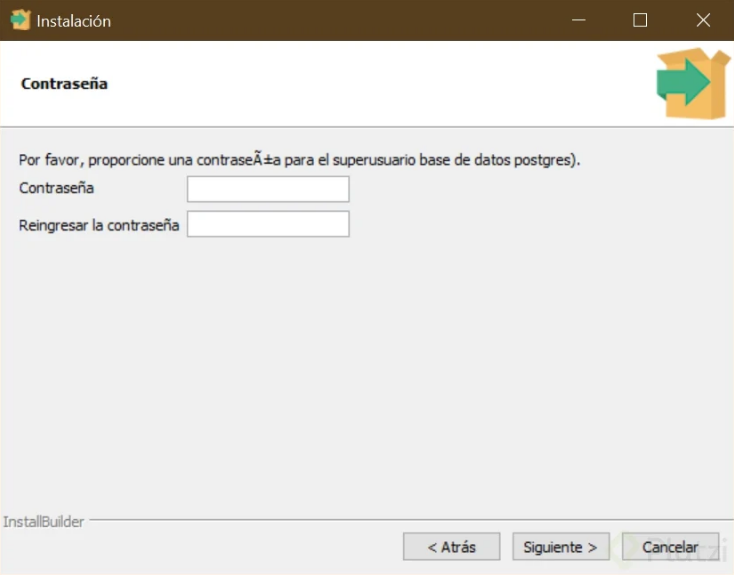


Ahora indicamos la carpeta donde irán guardados los datos de la base de datos, es diferente a la ruta de instalación del Motor de PostgreSQL, pero normalmente será una carpeta de nuestra carpeta de instalación. Puedes cambiar la ruta si quieres tener los datos en otra carpeta. Hacemos clic en siguiente.

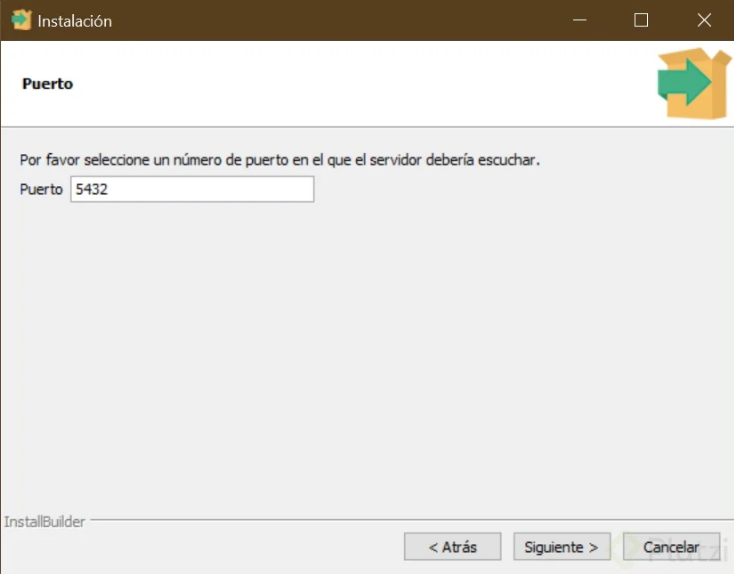


Ingresamos la **contraseña** del usuario **administrador**. De manera predeterminada, Postgres crea un usuario super administrador llamado **postgres** que tiene todos los permisos y acceso a toda la base de datos, tanto para consultarla como para modificarla. En este paso indicamos la clave de ese usuario super administrador.

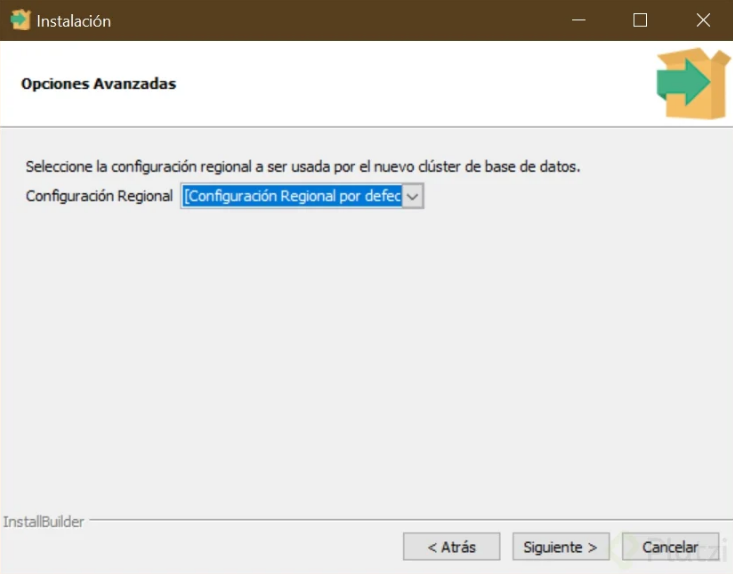
Debes ingresar una clave muy segura y guardarla porque la vas a necesitar después. Luego hacemos clic en siguiente.



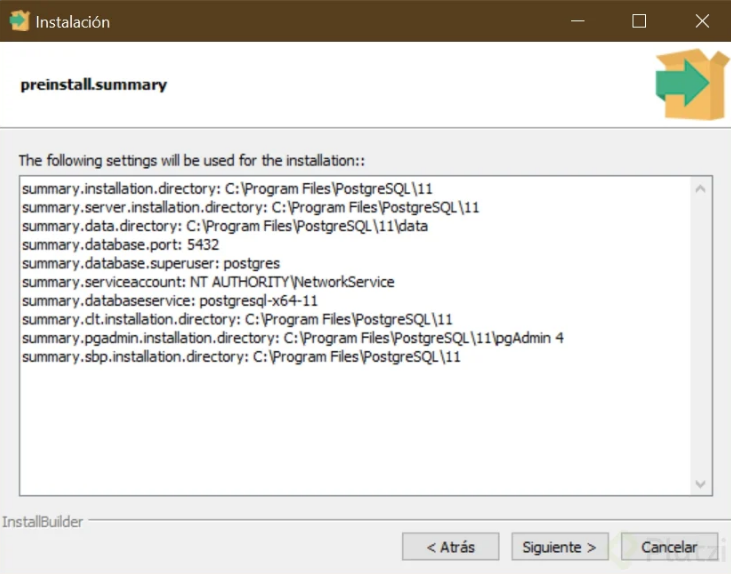
Ahora si queremos cambiar el puerto por donde el servicio de Postgresql estará escuchando peticiones, podemos hacerlo en la siguiente pantalla, si queremos dejar el predeterminado simplemente hacemos clic en siguiente.



La configuración regional puede ser la predeterminada, no es necesario cambiarla, incluso si vamos a usarla en español, ya que las tildes y las eñes estarán soportadas si dejas la configuración regional predeterminada. Es útil cambiarla cuando quieras dejar de soportar otras funciones de idiomas y lenguajes diferentes a uno específico. Luego hacemos clic en siguiente:



En pantalla aparecerá el resumen de lo que se va a instalar:



Al hacer clic en siguiente se muestra una pantalla que indica que PostgreSQL está listo para instalar, al hacer clic de nuevo en **siguiente** iniciará la instalación, espera un par de minutos hasta que la aplicación termine.

Una vez terminada la instalación, aparecerá en pantalla un mensaje mostrando que PostgreSQL ha sido instalado correctamente.

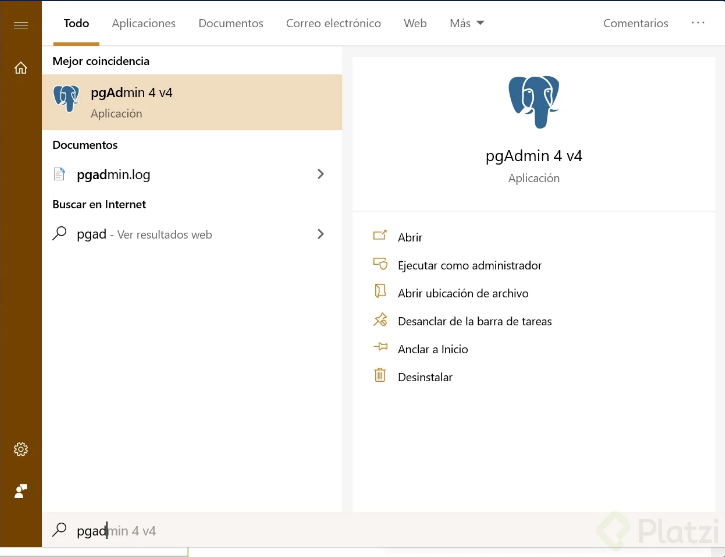


Podemos cerrar esta pantalla y proceder a comprobar que todo quedó instalado correctamente.

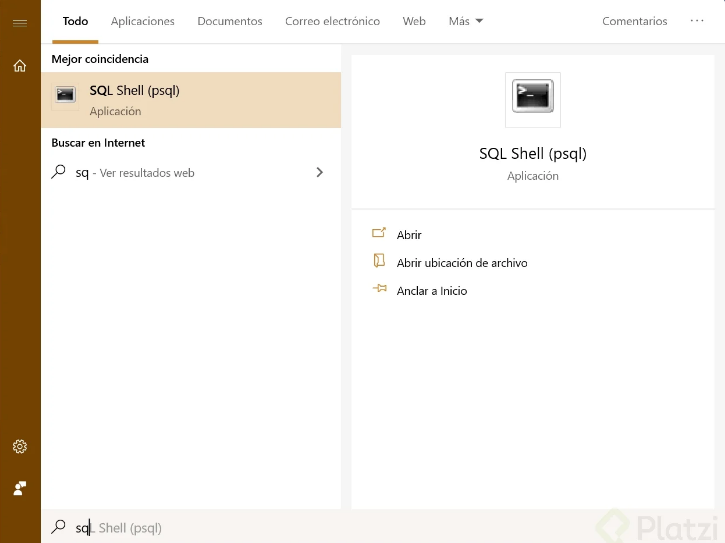
Vamos a buscar el programa **PgAdmin**, el cual usaremos como editor favorito para ejecutar en él todas las operaciones sobre nuestra base de datos.

También vamos a buscar la consola. Tanto la consola como PgAdmin son útiles para gestionar nuestra base de datos, una nos permite ingresar comando por comandos y la otra nos ofrece una interfaz visual fácil de entender para realizar todas las operaciones.

En el menú de Windows (o donde aparecen instalados todos los programas) buscamos “**PgAdmin**”



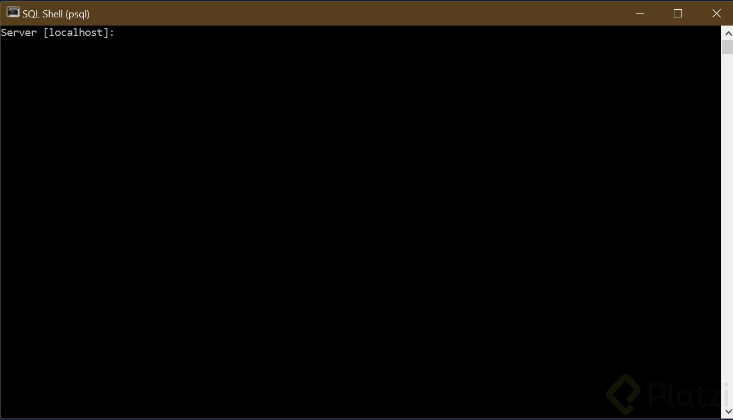
Ahora buscamos “**SQL Shell**”



Efectivamente, ahora aparecen las herramientas que vamos a utilizar.

Ahora vamos a crear una base de datos de prueba usando la consola y comprobaremos si existe usando **PgAdmin**, la crearemos para validar que la conexión con el servicio de base de datos interno funciona correctamente.

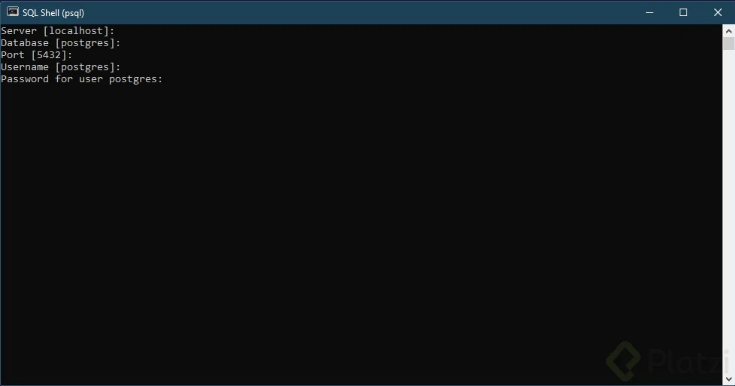
Para ello abrimos la consola, buscamos **SQL Shell** y lo ejecutamos. Veremos algo así:



Lo que vemos en pantalla es la consola esperando que ingresemos cada parámetro para la conexión.

Primero está el nombre del parámetro. En este caso es “**Server**” seguido de unos corchetes que contienen el valor predeterminado. Si presionamos “**Enter**” sin digitar nada la consola asumirá que te refieres al valor predeterminado, si en este caso presionamos “**Enter**” el valor asumido será “**Localhost**”. Localhost se refiere a nuestra propia máquina, si instalaste la base de datos en el mismo pc que estás usando para la consola, el valor correcto es **Localhost** o **127.0.0.1** (representan lo mismo).

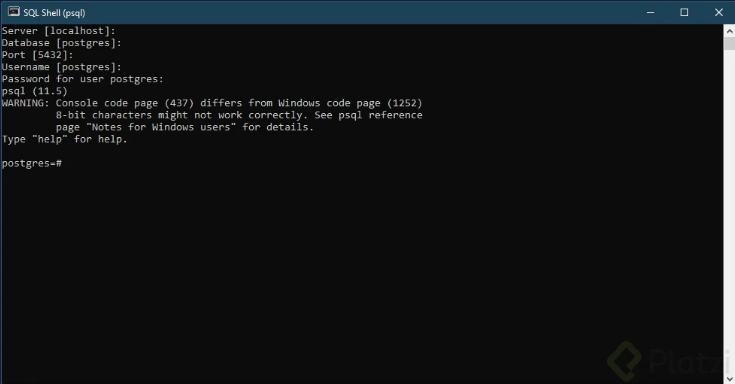
Podemos dejar todos los valores predeterminados (presionando “Enter”) hasta que la consola pregunte por la clave del usuario maestro:



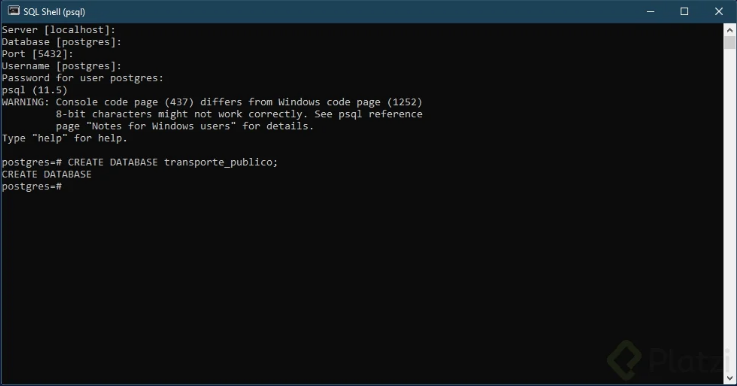
Debemos ingresar la **clave** que usamos cuando estábamos instalando Postgres, de lo contrario no podremos acceder. Presionamos **Enter** y veremos a continuación una pantalla que nos indica que estamos logueados en la base de datos y estamos listos para hacer modificaciones.

De manera predeterminada, la base de datos instalada es Postgres, la cual no debemos tocar, ya que ejecuta funciones propias del motor. Es usada por el Motor de PostgreSQL para interactuar con todas las bases de datos que vayamos a crear en el futuro.

La siguiente imagen indica que estamos conectados a la base de datos Postgres. Vamos a crear una base de datos nueva y luego saltar el cursor a esta base de datos recién creada.

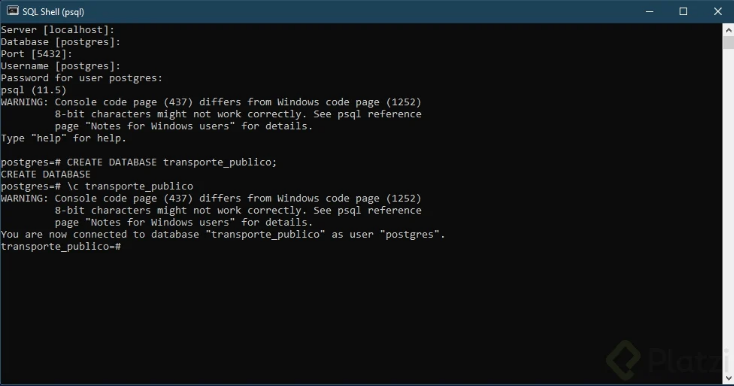


Para ello escribimos el comando “**CREATE DATABASE transporte\_publico**;” y presionamos “**Enter**”. Veremos:

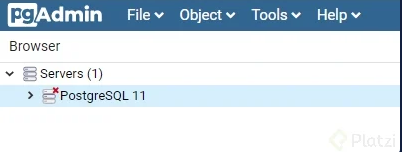


El mensaje “**CREATE DATABASE**” justo después de la línea que acabamos de escribir indica que la base de datos fue creada correctamente.

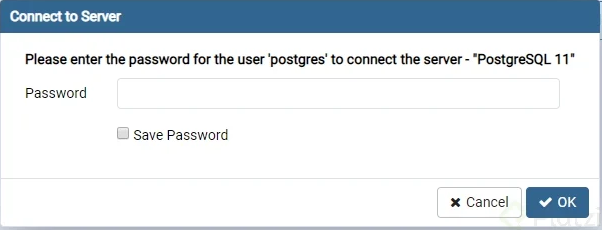
Para saltar a la base de datos recién creada ejecutamos el comando “**\c transporte\_publico**”, el cursor mostrará lo siguiente:



Ahora vamos a validar desde **PgAdmin** que la base de datos fue creada correctamente. Abrimos **PgAdmin** y nos encontramos con una lista de items a la izquierda, lo que significa que de manera predeterminada PgAdmin ha creado un acceso a nuestra base de datos local, el cual llamó “**PostgreSQL 13**”:

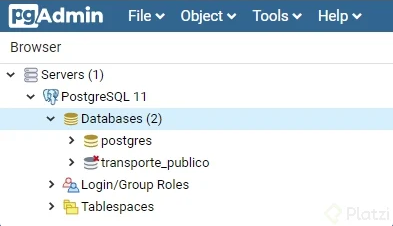


Al hacer doble clic sobre este elemento (“**PostgreSQL 13**”) nos pedirá ingresar la clave que hemos determinado para el super usuario **postgres**, al igual que la consola, hasta no ingresarla correctamente no nos podremos conectar:



Ingresamos la **clave**. Te recomiendo seleccionar la opción “**Save Password**” o “**Guardar Contraseña**”. Si la máquina sobre la que estás trabajando **es de confianza**, que seas **sólo tú** o tu equipo quien tenga acceso a ella, **de lo contrario**, no guardes la contraseña para mantenerla segura.

Veremos la lista de bases de datos disponibles, la predeterminada “**postgres**” y la que acabamos de crear usando la consola, lo que comprueba que la base de datos y la consola funcionan correctamente.

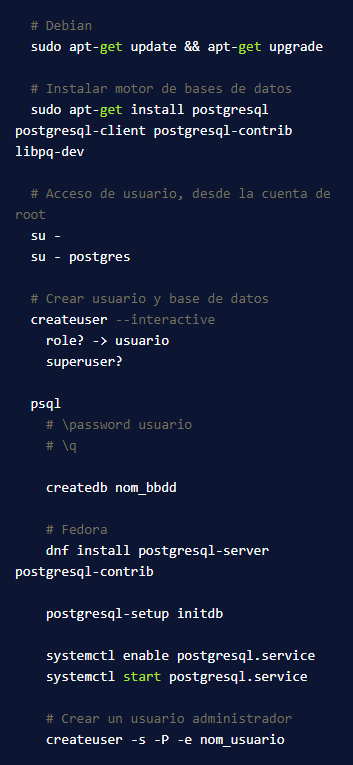


Ahora procedemos a eliminar la base de datos recién creada para comprobar que **PgAdmin** está correctamente configurada y si pueda realizar cambios sobre la base de datos.

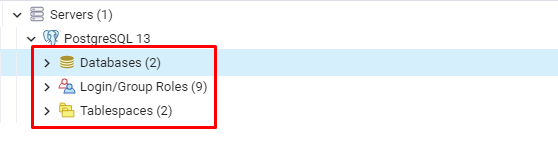
Para ello hacemos clic derecho sobre el elemento “**transporte\_publico**” y seleccionamos la opción “**Delete/Drop**”. Al mensaje de confirmar hacemos clic en **OK**.

Con ello, si el elemento “**transporte\_publico**” desaparece del menú de la izquierda comprobamos que PgAdmin funcionan correctamente.

# **Instalación de PostgreSQL en Linux**

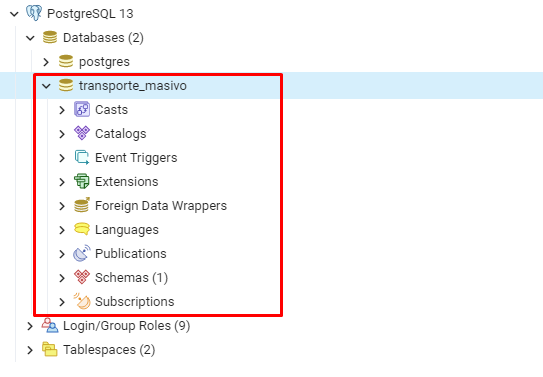


# **PgAdmin: Interacción con Postgres desde la Interfaz Gráfica**



En el primer nivel podemos encontrar:

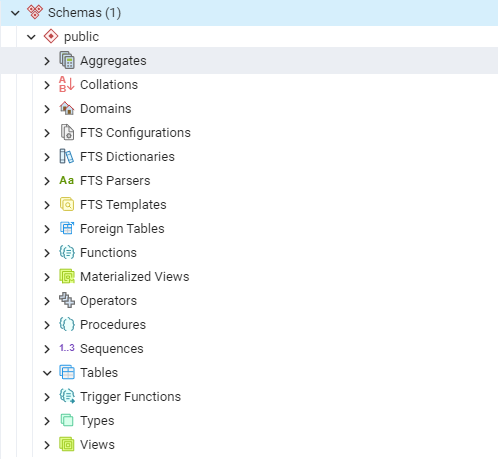
* **Databases**: Es donde se alojan todas las bases de datos.
* **Login/Group Roles**: Se encuentran todos los usuarios o grupos de usuarios que pueden acceder a la base de datos.
* **Tablespaces**: Es un espacio de memoria físico para guardar la información en la base de datos. Sí tu ordenador tiene varios discos, puedes configurarlo para que una tabla quede en un disco **C\:** y otra tabla quede en un disco llamado **D\:**



En el segundo nivel podemos encontrar:

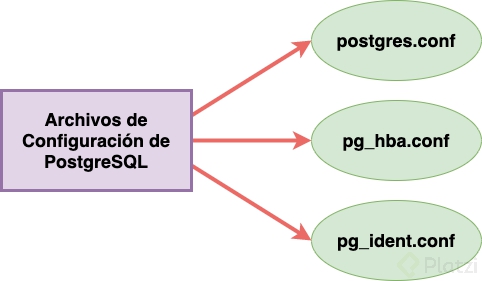
* **Casts**: Realiza la traducción de datos tipos explícitos
* **Catalogs**: Cataloga la información, es una especie de subdirectorios de tablas.
* **Event Triggers**: Ejecuta funciones en base a acciones que pasan en la base de datos.
* **Extensions**: Cuando se instalan módulos de terceros para extender la funcionalidad de la base de datos, se alojan aquí.
* **Foreingn Data Wrappers**: Acceder a bases de datos remoto.
* **Languages**: Aquí se puede tener tipos de datos diferentes, por ejemplo: Para que el motor acepte la Ñ o caracteres con acentos, etc.
* **Schemas**: Nos permite ver la información propia de las tablas.

En el tercer nivel podemos ver:



* **Public**: Es el schema predeterminado.
  + **Tables**: Es donde se encuentran todas las tablas de nuestra base de datos.

# **Archivos de Configuración**



* **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.

## **Replicación en PostgreSQL**

Uno de los métodos más usados para conseguir la Alta disponibilidad en nuestro servidor Postgres es implementar la Replicación. Con ello nos aseguramos que nuestro sistema esté activo el 99,9% del año. Replicas en “version and platform compatibility”.

## **¿Qué es la Replicación?**

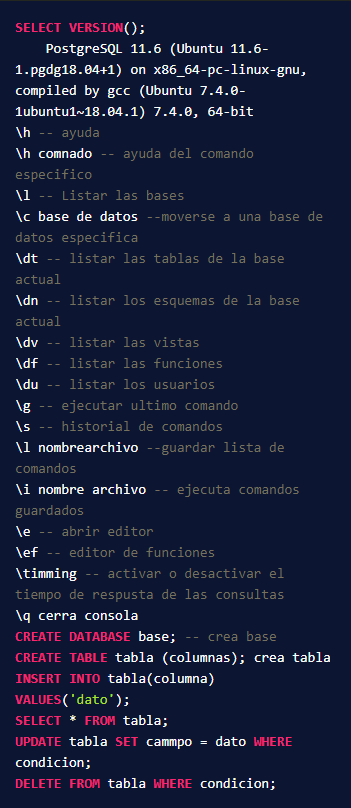
Es la transmisión de información derivada de las operaciones **DML** de una Base de Datos a otra, es decir, se transmite a otra Base de Datos las instrucciones **INSERT**, **UPDATE** y **DELETE** que se realicen en una Base de datos de forma que ambas BBDD tengan la misma información, así se obtiene una redundancia de datos.

# **Comandos más utilizados en PostgreSQL**

La **consola** en PostgreSQL es una herramienta muy potente para **crear**, **administrar** y **depurar** nuestra base de datos. podemos acceder a ella después de instalar PostgreSQL y haber seleccionado la opción de instalar la consola junto a la base de datos.

PostgreSQL está más estrechamente acoplado al entorno **UNIX** que algunos otros sistemas de bases de datos, utiliza las cuentas de usuario nativas para determinar quién se conecta a ella (de forma predeterminada). El programa que se ejecuta en la consola y que permite ejecutar consultas y comandos se llama **psql**, psql **es la terminal interactiva** para trabajar con PostgreSQL, es la interfaz de línea de comando o consola principal, así como **PgAdmin es la interfaz gráfica de usuario** principal de PostgreSQL.

## **Algunos comandos**



# **Tipos de datos**

## **Principales**

* Numéricos.
* Monetarios.
* Texto.
* Binarios.
* Fecha / Hora.
* Boolean.

## **Especiales**

* Geométricos.
* Dirección de Red.
* Texto tipo bit.
* XML, JSON.
* Arreglos.

Para más detalles de los tipos de datos que maneja postgres, [aquí](https://www.postgresql.org/docs/11/datatype.html) la documentación.

# **Jerarquía de Bases de Datos**

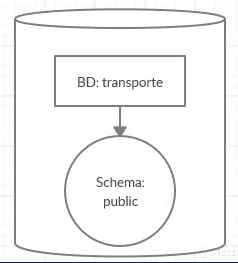
Toda jerarquía de base de datos se basa en los siguientes elementos:

* 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.

PostgreSQL es un motor de base de datos.

La estructura de la base de datos diseñada para el reto corresponde a los siguientes

elementos:



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

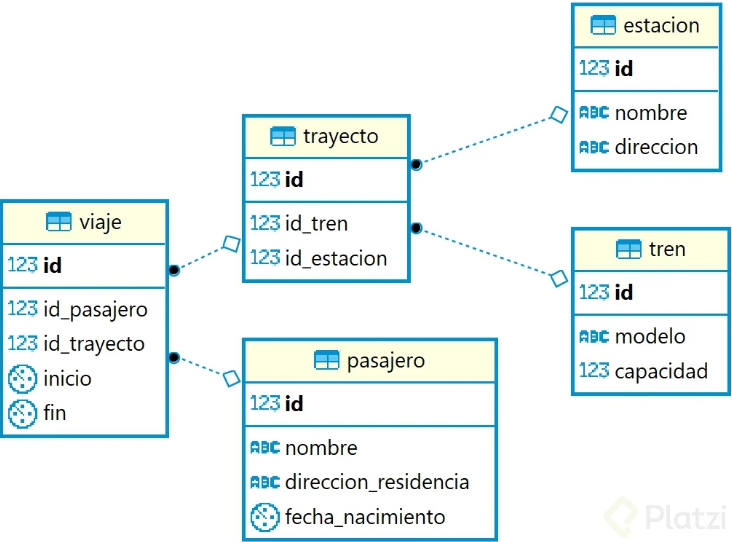
El esquema public contiene las siguientes tablas:

* Estación
* Pasajero
* Tren

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

* **Trayecto**
* **Viaje**

El esquema relacional entre las tablas corresponde al siguiente diagrama:



**Estación**

Contiene la información de las estaciones de nuestro sistema, incluye datos de nombre con tipo de dato texto y dirección con tipo de dato texto, junto con un número de identificación único por estación.

**Tren**

Almacena la información de los trenes de nuestro sistema, cada tren tiene un modelo con tipo de dato texto y una capacidad con tipo de dato numérico que representa la cantidad de personas que puede llevar ese tren, también tiene un **ID** único por tren.

**Trayecto**

Relaciona los trenes con las estaciones, simula ser las rutas que cada uno de los trenes pueden desarrollar entre las estaciones

**Pasajero**

Es la tabla que contiene la información de las personas que viajan en nuestro sistema de transporte masivo, sus columnas son nombre tipo de dato texto con el nombre completo de la persona, **direccion\_residencia** con tipo de dato texto que indica dónde vive la persona, **fecha\_nacimiento** tipo de dato texto y un **ID** único tipo de dato numérico para identificar a cada persona.

**Viaje**

Relaciona Trayecto con Pasajero ilustrando la dinámica entre los viajes que realizan las personas, los cuales parten de una estación y se hacen usando un tren.

# **Particiones**

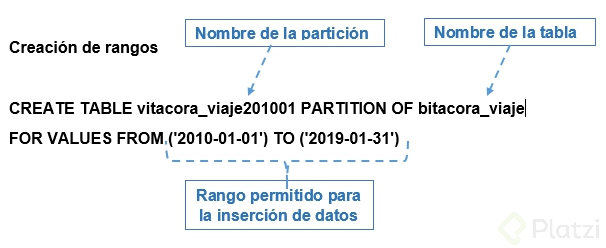
Es cuando hay mucha información en un table.

Dos factores importantes.

* Separación física de datos: Es posible guardar varias partes de la misma tabla en diferentes espacios de discos, incluso en otros discos.
* Estructura lógica: Funciona el mismo SELECT que veníamos haciendo todo el tiempo.

## **Como trabaja internamente**

Con el mismo nombre de la tabla asigna otras pequeñas tablas que tienen un rango definido.

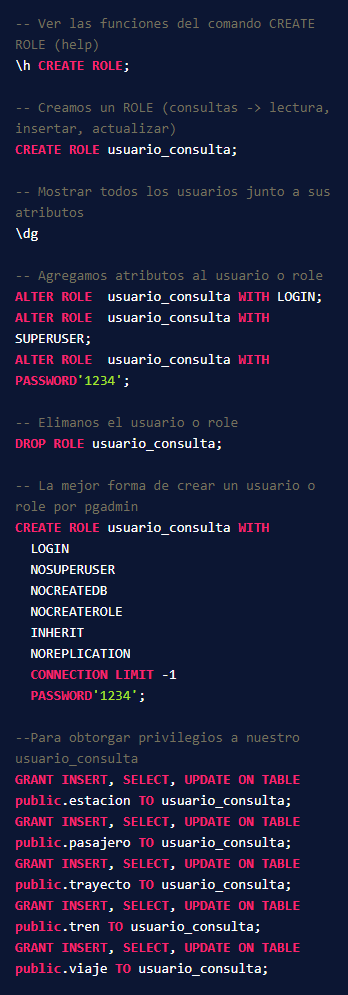


En las tablas particionadas no se hace uso de llaves primarias (primaryKey)

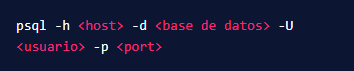
# **Creación de Roles**

Que puede hacer un ROLE

* Crear y Eliminar.
* Asignar atributos.
* Agrupar con otros roles.



**A los usuarios Linux**, no les sale la opción de configurar el inicio de **psql** con el usuario, todo eso se tiene que hacer a través de este comando.



## **Distintas formas de ver los privilegios**

Hay dos formas de ver los privilegios de un usuario en una tabla específica desde consola.



O



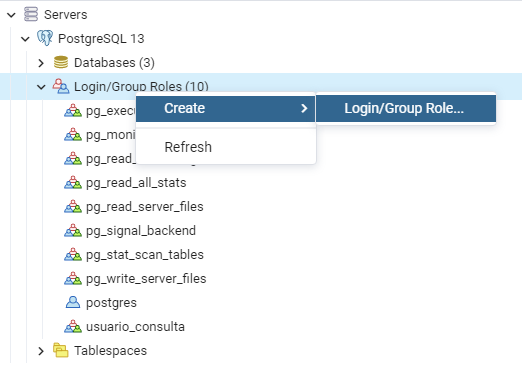
## **Crear roles**

Luego de la versión 9.3 de postgre los comandos para crear un nuevo rol son:

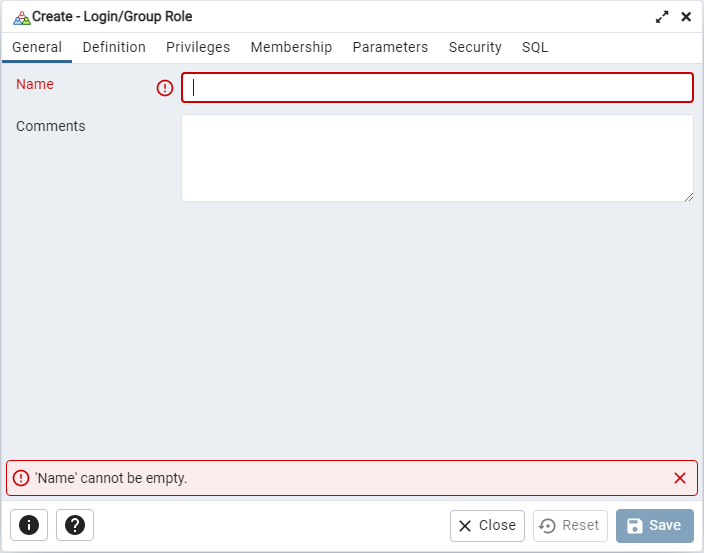
* CREATE USER
* CREATE ROL

Ambos son indistintos.

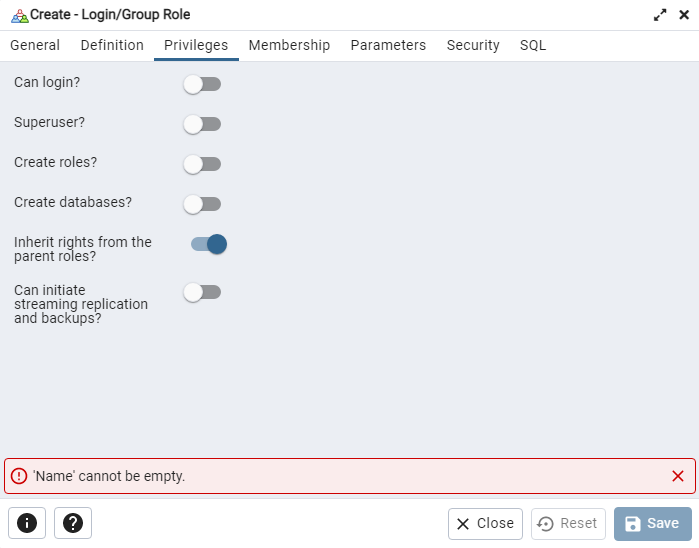
## **Creación de usuarios desde el administrador pgAdmin4**



Colocamos el nombre del usuario nuevo.



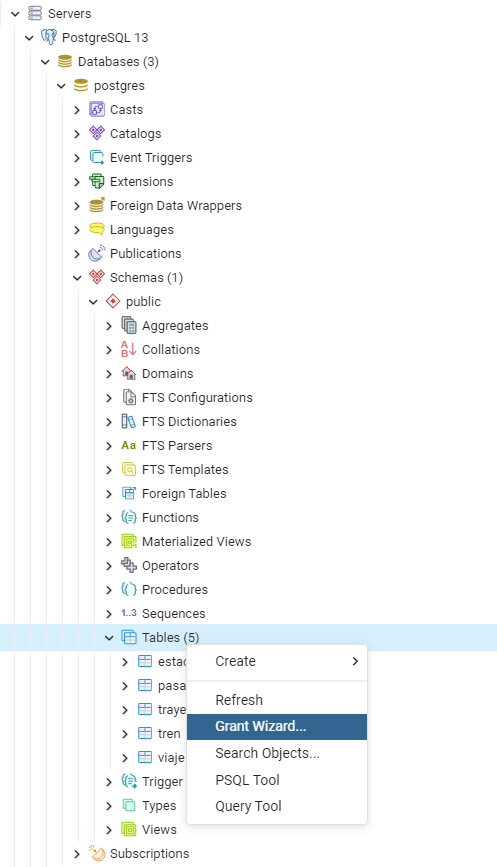
Nos dirigimos a la pestaña Privileges para setear los privilegios necesarios.



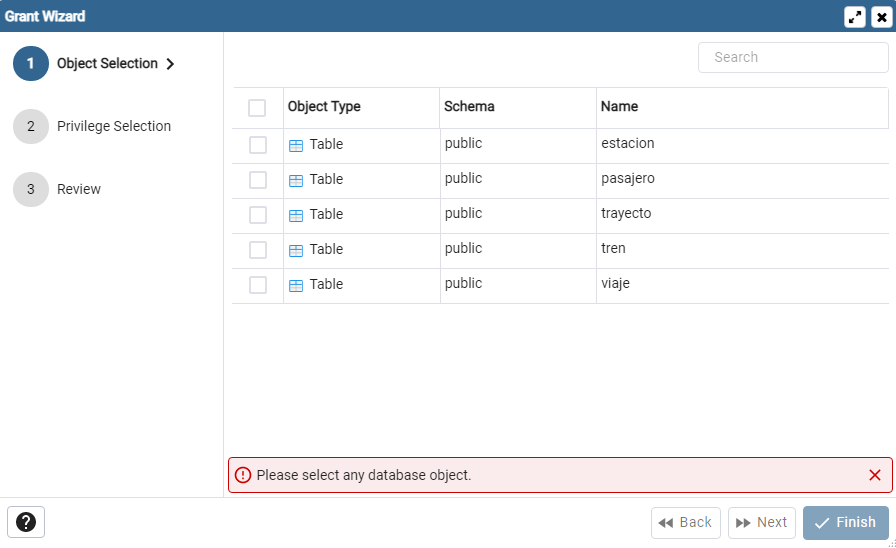
Una vez hecho esto, le damos click al botón Save para guardar los cambios.

## **Habilitar el nuevo usuario en pgAdmin4**

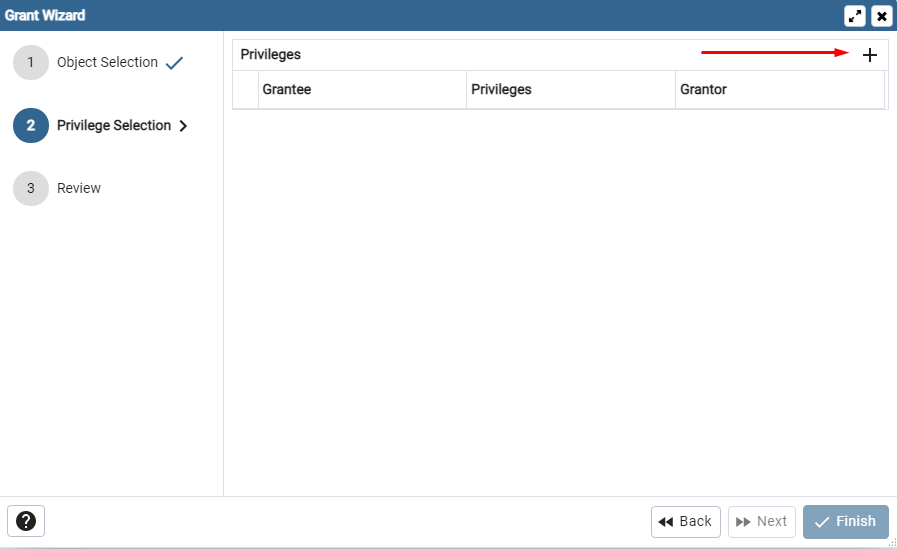
* Sobre **tables**, hacemos click derecho y elegimos la opción **Grant winzard...**



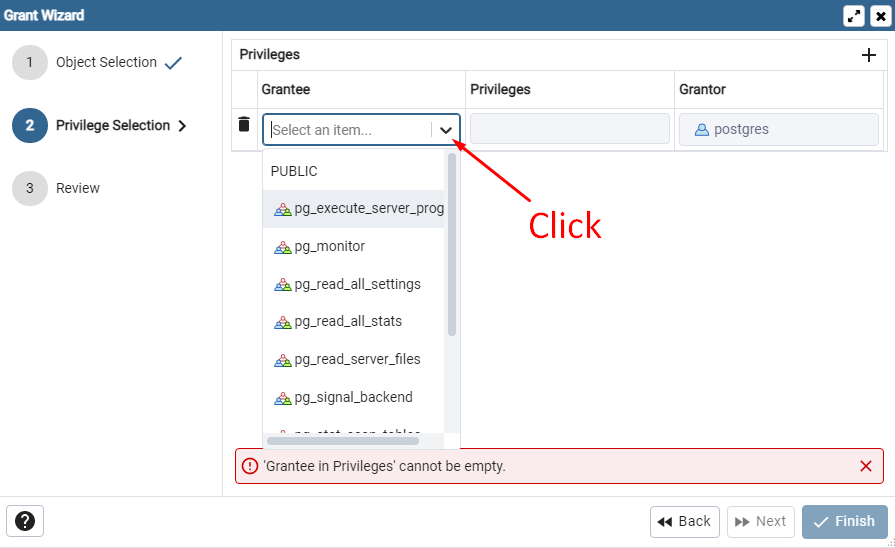
* Seleccionamos las tablas que deseamos que administre el **nuevo user**.



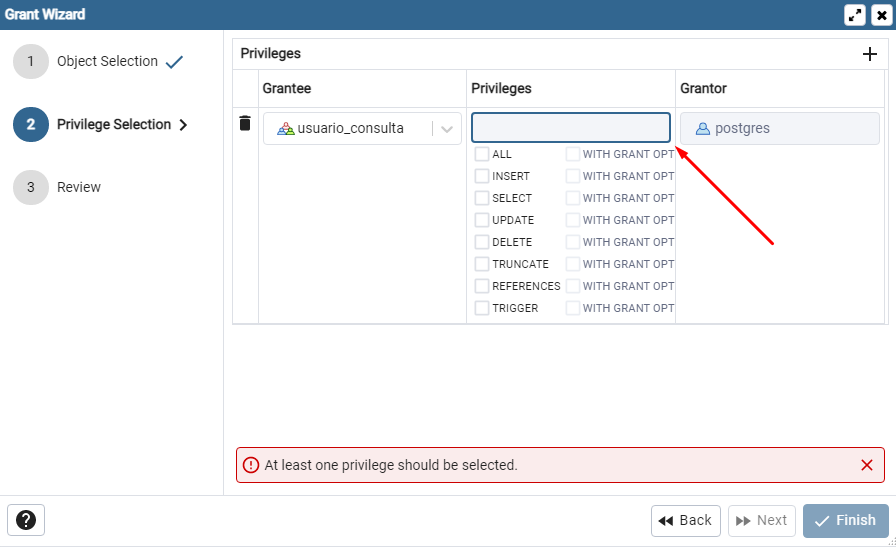
* Una vez seleccionadas, click en **Next**.
* Agregamos un nuevo privilegio.



* Click en el combo y elegimos el usuario.



* Click en **Privileges** y elegimos los privilegios del usuario.

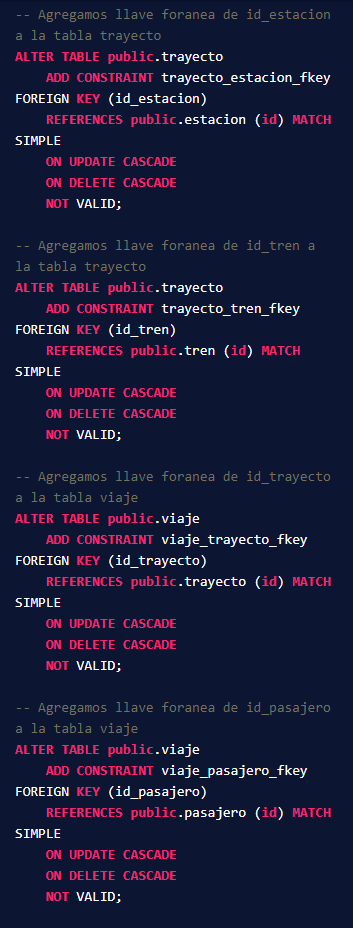


* Una vez elegidos los privilegios, click en **Next**.
* **pgAdmin4** nos mostrará un resumen de todo, **Finish** para finalizar el asistente.

# **Llaves foráneas**

Las llaves foráneas nos ayudan a tener:

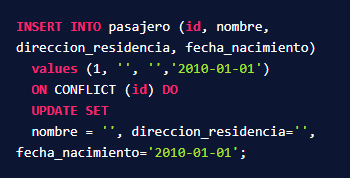
* Consistencia.
* Estructura.
* Tabla origen.
* Tabla destino.
* Acciones.



# **Funciones Especiales Principales**

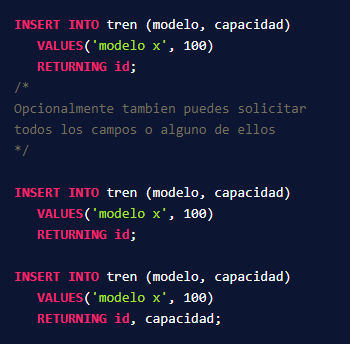
* **ON CONFLICT DO**: Esta instrucción nos permite especificar que debemos hacer en caso de un conflicto.

Ejemplo: Imaginamos que realizamos una consulta donde el **id** ya ha sido utilizado. Podemos especificar que, en ese caso, actualice los datos.



* **RETURNING**: Nos devuelve una consulta **select** de los campos sobre los que ha tenido efecto la instrucción.

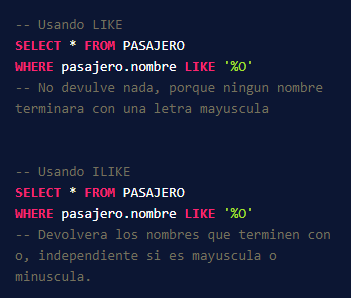
Ejemplo: Queremos saber cuál es el **id** que le fue asignado a un dato insertado.



* **LIKE / ILIKE**: Las funciones **like** y **ilike** sirven para crear consultas a base de expresiones regulares.

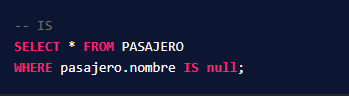
**Like** considera mayúsculas y minúsculas, mientras que **ilike** solo considera las letras.

Ejemplo: Busquemos a los pasajeros con nombre que terminen con la letra **o**

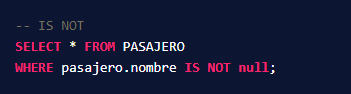


* **IS / IS NOT**: Permite hacer comprobación de valores especiales como **null**

Ejemplo: Consultemos a todos los usuarios que tengan como **direccion\_residencia NULL**

****

Ahora a los que si tengan la **direccion\_recidencia** con algún valor.



# **Funciones Especiales Avanzadas**

Estas son las funciones avanzadas más comunes en postgres.

* **COALESCE**: Te permite comparar 2 valores y retornar el que no es nulo. Pero la función acepta un número ilimitado de argumentos. Cuando son más de 2 argumentos devuelve el primero no **NULL** (en sentido de izquierda a derecha). Si todos los argumentos son NULL, devuelve NULL.

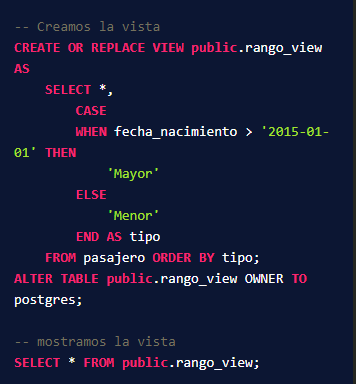
Se utiliza para evitar devolver valores nulos. Esto es útil cuando el valor devuelto tiene que ser usado dentro de una función.

* **NULLIF**: Te permite comparar 2 valores y retorna NULL si son iguales. Si no son iguales retorna el argumento de la izquierda. Esta función se puede usar como denominador de una división con el argumento de la izquierda igual a cero. La división siempre se dividirá por el argumento de la izquierda salvo cuando sea cero. Para ese caso NULLIF devuelve NULL, haciendo que el resultado de la división sea NULL y no indefinido.
* **GREATEST**: Te permite comparar un arreglo de valores y te retorna el mayor
* **LEAST**: Idéntico anterior, pero para este caso retorna el menor.
* **BLOQUES ANÓNIMOS**: Al igual que el desarrollo de software te permite ingresar condicionales, pero dentro de una consulta de base de datos. Se agrega más información a una misma tabla sin agregar ningún tipo de consulta.

# **Vistas**

Tipos de vistas:

* **Vista Volátil**: Consulta con data actualizada.
* **Vista Materializada**: Consulta con data persistente.

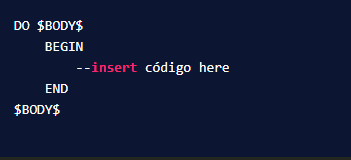




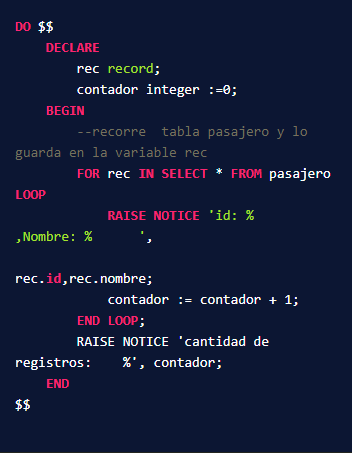
# **PL/SQL**

* **PL** (**P**rocedural **L**anguage), también conocido como **procedimientos almacenados**, estas nos ayudan a desarrollar código directamente en el motor de bases de datos.
* Estructura de un **Pl** es: Declaración + uso de variable+ código +fin + retorno de valores o no retorna valores. Un bloque de código se ejecuta con la palabra **DO $$ BEGIN** --insert código aquí **END $$**
* **RAISE NOTICE** ‘message’, esta sentencia es para enviar un mensaje en el log de postgres.
* Retornar una tabla.

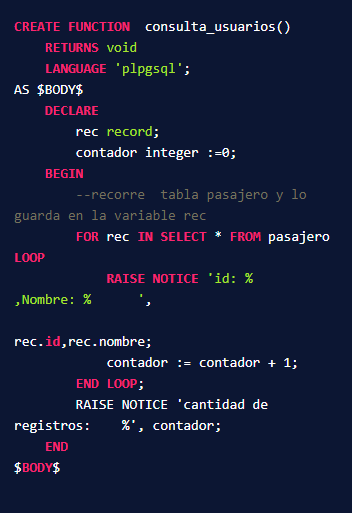
DO$$ -Declaración de un bloque de código SQL



Ejemplo de declaración de bloques de código con plpgsql.

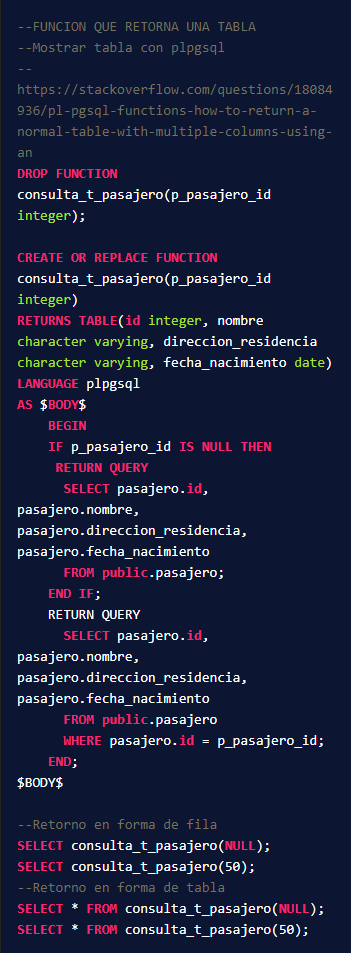


**CREATE FUNTION** - Declaración de una función SQL.



**Retornar una tabla con plpgsql**

Es **importante** cual **select** uses para llamar la función. La función funciona de la siguiente manera en el parámetro sí se introduce **NULL** retorna toda la lista, si se introduce **id** retornará esa tupla.



# **Triggers**

## **¿Que son los Triggers?**

Los triggers, al igual que las tablas, son objetos en los cuales podemos programar unas instrucciones con el fin de que se **disparen** (he ahí su nombre) cuando haya un evento (cambio) en la tabla relacionada. **No puede haber trigger sin ser relacionada a una tabla**.

Los cambios que puede tener una tabla pueden ser (**update**, **insert**, **delete**).

## **¿Para qué necesitamos los triggers?**

Un ejemplo claro sería, que, como administrador de la base de datos, necesitamos tener un control y saber que usuarios han estado interactuando con la base de datos disparando dichos eventos.

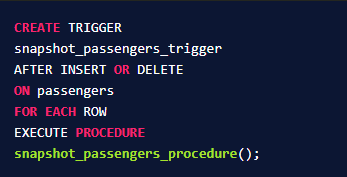
## **¿Qué acciones puede realizar un trigger?**

Un trigger puede dispararse ya sea **antes**, **después** o **en lugar de**, de un evento. Por ejemplo: Un usuario al ingresar datos a una tabla por medio del comando **Insert**, el trigger puede ejecutarse justo antes almacenando en otra tabla información relacionada sobre:

* ¿Quién lo hizo?
* La hora.
* Cantidad de datos agregados, etc. Y esto se hará de forma automática.

También se puede por ejemplo antes de ejecutar el comando **update**, hacer una copia de respaldo.

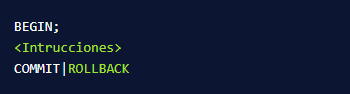
Usualmente los triggers se relacionan con el mantenimiento y administración de las bases de datos.



# **Transacciones**

Las transacciones, tienen la capacidad para empaquetar varios pasos en una sola operación “todo o nada” y si ocurre alguna falla que impida que se complete la transacción, entonces ninguno de los pasos se ejecuta y no se afecta la base de datos en absoluto.

Las transacciones tienen la siguiente estructura postgres.



Postgres en las operaciones normales usa de manera implícita el **rollback**.