
SENPAI
ACADEMY

Base de Datos

SENPAI
ACADEMY

Base de Datos - PostgreSQL

Como vimos en la clase anterior, podemos utilizar el programa PGAdmin para crear Bases de Datos, crear o manipular tablas, realizar consultas, etc.

También es bastante común que queramos acceder a la Base de Datos a través de una terminal, para lo cual primero debemos agregar Postgres a las variables de entorno, de forma de que podamos utilizar luego el comando “psql” en la terminal.

- Instrucciones para Windows
- Instrucciones para Mac

SENPAI
ACADEMY

PostgreSQL - Creación de Tablas

Vimos el uso del comando ***CREATE TABLE*** para crear una nueva tabla en la Base de Datos, y recuerden que pueden encontrar todos los tipos de datos acá

```
CREATE TABLE nombre_tabla (
    Nombre de la Columna + Tipo de Dato + Restricciones (si tiene)
)
```

```
CREATE TABLE person (
    id int,
    first_name VARCHAR (50),
    last_name VARCHAR (50),
    gender VARCHAR(6),
    date_of_birth date
)
```

Acá estamos limitando que first_name y last_name pueden tener como largo 50 caracteres, y gender solo 6.

SENPAI ACADEMY

PostgreSQL - Creación de Tablas

Normalmente vamos a querer crear nuestras tablas con restricciones (“constraints”) para poder asegurar la integridad de los datos que se guardan en ellas.

```
CREATE TABLE person (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    first_name VARCHAR (50) NOT NULL,
    last_name VARCHAR (50) NOT NULL,
    gender VARCHAR(6) NOT NULL,
    address VARCHAR(150),
    date_of_birth date NOT NULL
);
```

BIGSERIAL es un int que se incrementa automáticamente, lo cual es muy útil para los IDs.

La clave primaria (**PRIMARY KEY**) es la clave que identificará cada uno de los registros de la base de datos de forma única. A veces podríamos utilizar un dato de la persona, como la cédula, o número de pasaporte, pero es una práctica muy común tener una columna con un ID autoincremental.

SENPAI ACADEMY

PostgreSQL - Insertar datos

Para agregar datos a una tabla, utilizamos el comando ***INSERT***
Como la tabla que creamos tiene un id que se autoincrementa, no es necesario proporcionar uno en los datos.

```
INSERT INTO person (
  first_name,
  last_name,
  gender,
  date_of_birth
)
VALUES ('John', 'Smith', 'MALE', '1990-05-10');

INSERT INTO person (
  first_name,
  last_name,
  gender,
  address,
  date_of_birth
)
VALUES ('Anne', 'Lewis', 'FEMALE', 'St Barbara 5442', '1990-05-10');
```

```
3  SELECT * FROM person;
```

Salida de Datos | Explain | Mensajes | Notificaciones

	id [PK] bigint	first_name character varying (50)	last_name character varying (50)	gender character varying (6)	address character varying (150)	date_of_birth date
1	1	John	Smith	MALE	[null]	1990-05-10
2	2	Anne	Lewis	FEMALE	St Barbara 5442	1990-05-10

SENPAI ACADEMY

PostgreSQL - Insertar datos

Como en la tabla person pusimos determinadas restricciones, la Base de Datos no nos permite agregar un nuevo registro con datos incompletos:

Query Editor | Historial de Consu...

```
1  INSERT INTO person (
2    first_name,
3    last_name
4  ) VALUES ('Dean', 'Thomas')
5
6
```

Salida de Datos | Explain | Mensajes | Notificaciones

```
ERROR:  null value in column "gender" of relation "person" violates not-null constraint
DETAIL:  Failing row contains (3, Dean, Thomas, null, null, null).
Estado SQL: 23502
```

SENPAI ACADEMY

PostgreSQL - Consultar datos

A la base de datos le podemos realizar consultas, peticiones, que normalmente vamos a llamar ***query,*** y estas nos permiten leer ciertos datos. En una query podemos enviar determinados parámetros para filtrar los datos, ordenarlos, agruparlos, etc.

Podemos usar el comodín ***** para traer todas las columnas, o podemos especificar cuales queremos.

Podemos ordenar nuestros resultados con ***ORDER BY*** por una columna, o más (por defecto es **ASC** ascendente, pero si ponemos **DESC** lo hará de forma descendente).

Podemos filtrar los resultados con ***WHERE*** indicando la condición que queremos que cumplan.

```sql
SELECT * FROM person;

SELECT first_name, last_name FROM person;

SELECT * FROM person ORDER BY date_of_birth;
SELECT * FROM person ORDER BY date_of_birth DESC;

SELECT * FROM person WHERE gender = 'Male' ORDER BY date_of_birth;
```

SENPAI ACADEMY

PostgreSQL - Consultar datos

En la cláusula del WHERE, donde indicamos las condiciones que queremos que los resultados de la query cumplan, podemos realizar filtros muy complejos, ayudándonos con estos operadores:

- **AND** , **OR**: para unir más de una condición
- **>**, **<**, **>=**, **<=**, **<>**: Mayor, Menor, Mayor igual, Menor igual, Distinto a
- **IN**: para validar que pertenezcan a una lista
- **BETWEEN** {value 1} **AND** {value 2}: para filtrar que el valor de una columna esté entre 2 valores
- **LIKE**, **ILIKE**: Sirve para realizar búsquedas que mapeen determinados patrones, por ejemplo, buscar todos los correos que tengan de dominio “@gmail.com”:
  ```
  SELECT * FROM person WHERE email like '%@gmail.com'
  ```

  En este operador, el % funciona como un comodín que dice “cualquier carácter”, podríamos ponerlo también al final:
  ```
  SELECT * FROM person WHERE email like '%@gmail%'
  ```

  ILIKE ignora las mayúsculas y minúsculas.

SENPAI ACADEMY

PostgreSQL - Ejercicio

- Crear una Base de Datos de prueba, o utilizar una que tengan y crear la tabla “person” con *id* autoincremental, *first_name*, *last_name*, *gender*, *address*, *date_of_birth*, y *country*.
- Utilizar el script “Insert 1000 person.sql” del Campus, para agregar 1000 datos a la tabla.
- Escribir las consultas que resuelvan los siguientes casos:

1. Obtener todos los paises distintos (Investigar cómo utilizar la palabra ***DISTINCT***)
2. Obtener todas las personas que nacieron en ‘China’ o ‘Russia’.
3. Obtener todas las personas que nacieron en ‘China’ o ‘Brazil’, y además nacieron luego de 1990.
4. Obtener todas las personas que pertenecen a una lista de países (por lo menos 3) utilizando el operador **IN**.
5. Obtener todas las personas que nacieron entre 1990 y 2000.
6. Obtener todas las personas que en su primer nombre contienen los caracteres ‘ton’.

SENPAI ACADEMY

PostgreSQL - Eliminar datos

Para borrar registros vamos a utilizar el comando ***DELETE***, y es muy importante que siempre recordemos agregar una cláusula ***WHERE*** a esta query (anoser que queramos borrar todos los registros de una tabla).

Lo normal es que utilicemos la columna que sea ***PRIMARY_KEY*** en la tabla para identificar que registro queremos borrar, pero también podríamos querer eliminar todos los registros de determinado país por ejemplo.

```
DELETE FROM person WHERE id = 3;

DELETE FROM person WHERE country = 'Sweden';
```

SENPAI ACADEMY

PostgreSQL - Actualizar datos

Para actualizar registros vamos a utilizar el comando ***UPDATE***, y normalmente también vamos a utilizar una cláusula ***WHERE*** con esta query para indicar qué fila o filas exactamente queremos actualizar, pero si no ponemos ***WHERE*** vamos a estar actualizando todas las filas de la tabla (Utilizar con cuidado).

Con ***UPDATE*** tenemos que indicar qué tabla queremos actualizar, y luego con ***SET*** indicar qué columnas queremos cambiar con qué valores, separando con una coma cada columna que queremos actualizar:

```
UPDATE person
SET country = 'Francia', address = 'Paris'
WHERE country = 'France';
```

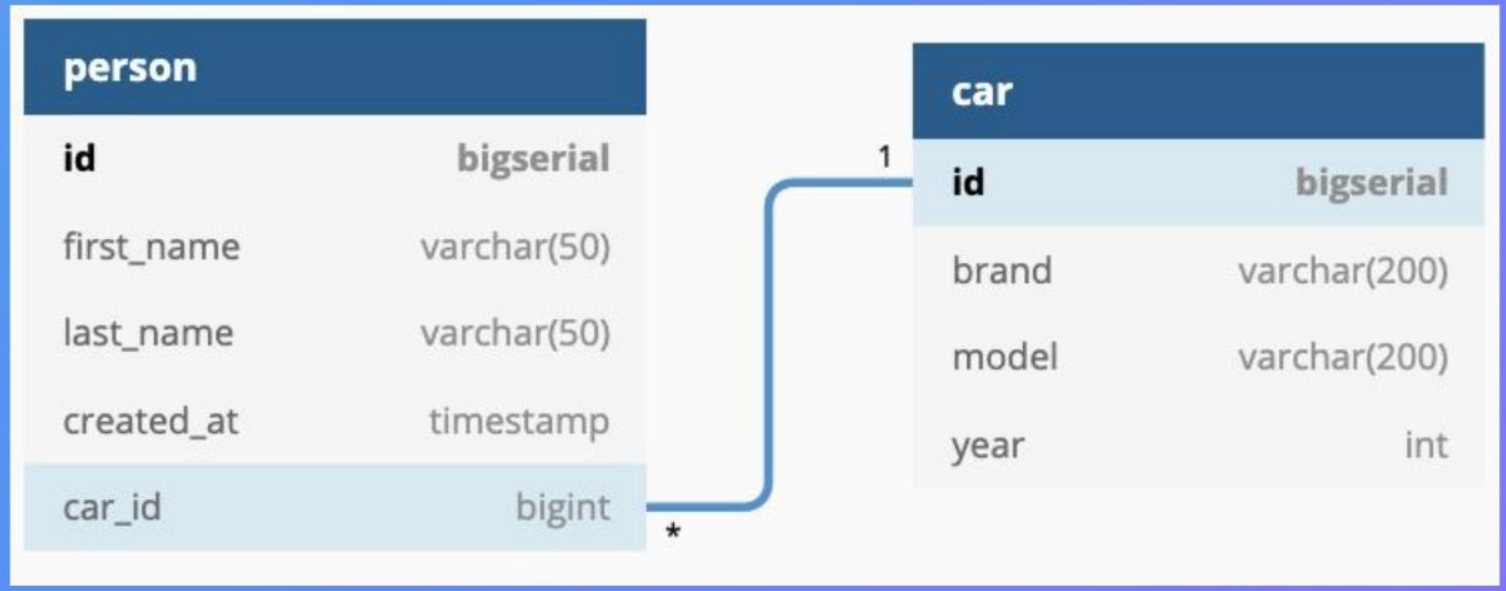
SENPAI ACADEMY

PostgreSQL - Relaciones, Claves Foráneas

Como ya mencionamos, Postgres es una Base de Datos relacional, por lo que vamos a poder declarar relaciones entre dos entidades, por ejemplo, Persona y Auto.

Para esto se utilizan las "***Foreign Keys***" o claves foráneas. Esta clave es una columna que hace referencia a la clave primaria de otra tabla.

En el siguiente ejemplo, ***`car_id`*** en la tabla **person** es una clave foránea con la tabla **car**.



SENPAI ACADEMY

PostgreSQL - Relaciones, Claves Foráneas

Cuando creamos una tabla, podemos indicar las claves foráneas con "***REFERENCES***" indicando la tabla, y la columna de la tabla a la que queremos hacer referencia.

En este ejemplo también estamos agregando una restricción ***UNIQUE*** para que un auto no pueda pertenecer a dos personas.

```sql
create table car (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    brand VARCHAR(100) NOT NULL,
    model VARCHAR(100) NOT NULL,
    year INT NOT NULL
);

create table person (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    gender VARCHAR(7) NOT NULL,
    email VARCHAR(100),
    date_of_birth DATE NOT NULL,
    country VARCHAR(50) NOT NULL,
    car_id BIGINT REFERENCES car (id),
    UNIQUE(car_id)
);
```

SENPAI ACADEMY

PostgreSQL - INNER JOIN

El ***INNER JOIN*** es utilizado para combinar dos tablas que tengan una relación. Si tomamos la tabla A y B, y hay una clave foránea que está presente en las dos tablas, el ***INNER JOIN*** nos devuelve ese resultado.

Al poner ***FROM person p*** estamos dando un "alias" ***p*** a la tabla person para hacer referencia a ella en el ***ON*** de el ***JOIN***, y en el ***SELECT***, sin necesidad de escribir todo el nombre

```
SELECT p.first_name, p.last_name, c.*
FROM person p
    INNER JOIN car c ON c.id = p.car_id;
```

Salida de Datos | Explain | Mensajes | Notificaciones

	first_name character varying (50)	last_name character varying (50)	id bigint	brand character varying (100)	model character varying (100)	year integer
1	Omar	Colmore	1	Land Rover	Sterling	2015

Si solo utilizamos **JOIN** es lo mismo que **INNER JOIN**

```
SELECT *
FROM person p
    JOIN car c on c.id = p.car_id;
```

SENPAI ACADEMY

PostgreSQL - LEFT JOIN

El ***LEFT JOIN*** es utilizado para combinar dos tablas que tengan una relación, pero también ver los resultados que no están presentes en las dos tablas. Si tomamos la tabla A y B, y hay una clave foránea que está presente en las dos tablas, el ***LEFT JOIN*** nos devuelve los resultados de la tabla A y para los que tienen datos en la tabla B, también trae esos datos.

```
SELECT p.first_name, p.last_name, c.*
FROM person p
    LEFT JOIN car c ON c.id = p.car_id;
```

Salida de Datos | Explain | Mensajes | Notificaciones

	first_name character varying (50)	last_name character varying (50)	id bigint	brand character varying (100)	model character varying (100)	year integer
1	Omar	Colmore	1	Land Rover	Sterling	2015
2	John	Matuschek	[null]	[null]	[null]	[null]
3	Fernanda	Beardon	[null]	[null]	[null]	[null]

SENPAI ACADEMY

Base de Datos - Siguientes Pasos

La semana que viene vamos a ver cómo comunicarnos con la Base de Datos desde nuestro servidor en NodeJS, por lo que ya pueden ir creando su Base de Datos con los tipos de datos correctos, las relaciones correctas, y hasta insertar datos de prueba.

Pueden usar la página mockaroo para generar hasta mil INSERTS para sus tablas.

SENPAI
ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE