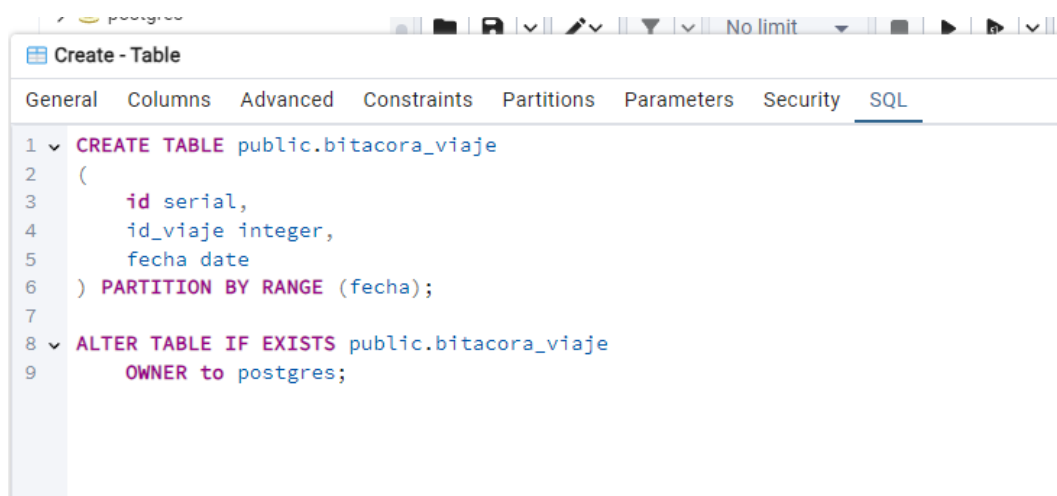


Elementos de una tabla

Particiones

Una de las ventajas de las tablas particionadas es que

- Las particiones consisten en la separación física de datos conservando la estructura lógica, esto ayuda a acelerar nuestras consultas
- Si la tabla que vamos a crear es particionada debemos indicarlo en la pestaña general
- No es posible crear llaves primarias en tablas particionadas



The screenshot shows a 'Create - Table' dialog box with the 'SQL' tab selected. The SQL code is as follows:

```
1 CREATE TABLE public.bitacora_viaje
2 (
3     id serial,
4     id_viaje integer,
5     fecha date
6 ) PARTITION BY RANGE (fecha);
7
8 ALTER TABLE IF EXISTS public.bitacora_viaje
9     OWNER to postgres;
```

La siguiente imagen muestra un ejemplo de como se particionaría una tabla en varias tablas pequeñas que contendrán diferentes rangos de fecha.

Tal y como esta configurada actualmente la tabla no funcionaría debido a que no se le ha asignado ningún particionamiento en memoria.

The screenshot shows a database query editor with a 'Query' tab. The query is:

```
1 INSERT INTO public.bitacora_viaje( id_viaje, fecha)
2 VALUES (1, '2010-01-01');
```

The 'Messages' tab is selected, showing the following error:

```
ERROR: La llave de particionamiento de la fila que falla contiene (fecha) = (2010-01-01).no se encontró una partición de «bitacora_viaje» para el registro

ERROR: no se encontró una partición de «bitacora_viaje» para el registro
SQL state: 23514
Detail: La llave de particionamiento de la fila que falla contiene (fecha) = (2010-01-01).
```

Para completar esto, asignaremos el particionamiento de la siguiente forma:

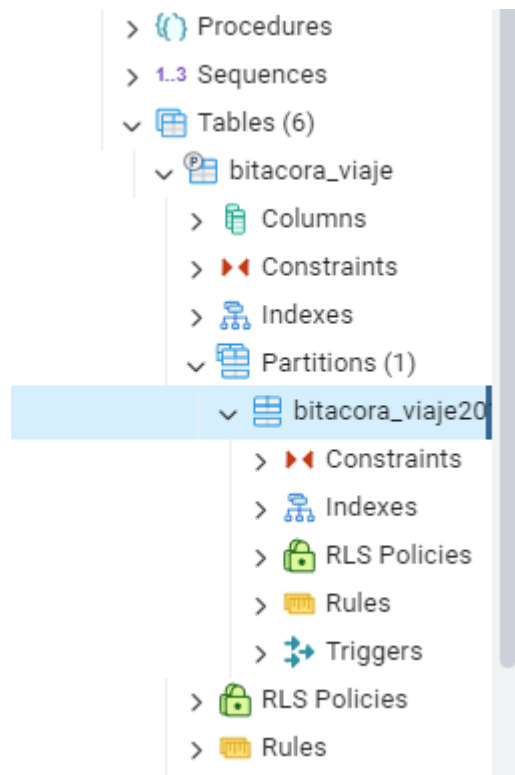
The screenshot shows a database query editor with a 'Query' tab. The query is:

```
1 CREATE TABLE bitacora_viaje201001 PARTITION OF bitacora_viaje
2 FOR VALUES FROM ('2010-01-01') TO ('2019-01-01');
3 INSERT INTO public.bitacora_viaje( id_viaje, fecha)
4 VALUES (1, '2010-01-01');
5
6 SELECT * FROM bitacora_viaje;
```

The 'Data Output' tab is selected, showing the following table:

	id integer	id_viaje integer	fecha date
1	2	1	2010-01-01

Si nos vamos al apartado de partitions, vemos que se ha creado correctamente:



Es muy importante que para este ejemplo se tenga en cuenta que, la partición permitirá insertar datos que se encuentren entre las fechas establecidas.



No es posible crear claves primarias en tablas particionadas.

Roles

Que puede hacer un ROLE:

- **Crear y Eliminar**
- **Asignar atributos**
- **Agrupar con otros roles**
- **Roles predeterminados**

```
Query  Query History
1  CREATE ROLE usuario_consulta;
```

Para ver la lista de roles, ejecutamos el siguiente comando en consola:

```
transporte=#
transporte=# \dg

                Lista de roles
Nombre de rol | Atributos
-----+-----
postgres      | Superusuario, Crear rol, Crear BD, Replicación, Ignora RLS
usuario_consulta | No puede conectarse

transporte=#
```

Vemos que el rol asignado no puede acceder a la base de datos. Tenemos que modificar lo siguiente:

```
transporte=# ALTER ROLE usuario_consulta WITH LOGIN;
ALTER ROLE
transporte=# \dg

                Lista de roles
Nombre de rol | Atributos
-----+-----
postgres      | Superusuario, Crear rol, Crear BD, Replicación, Ignora RLS
usuario_consulta |

transporte=#
```

Para asignar permisos al rol, consultamos la lista de permisos que hay y asignaremos por ejemplo el de superusuario:

```
transporte=# ALTER ROLE usuario_consulta WITH SUPERUSER;
ALTER ROLE
transporte=# \dg

                Lista de roles
Nombre de rol | Atributos
-----+-----
postgres      | Superusuario, Crear rol, Crear BD, Replicación, Ignora RLS
usuario_consulta | Superusuario

transporte=#
```

También tendremos que asignar una contraseña a nuestro usuario:

```
transporte=# ALTER ROLE usuario_consulta WITH PASSWORD '1234';
ALTER ROLE
transporte=#
```

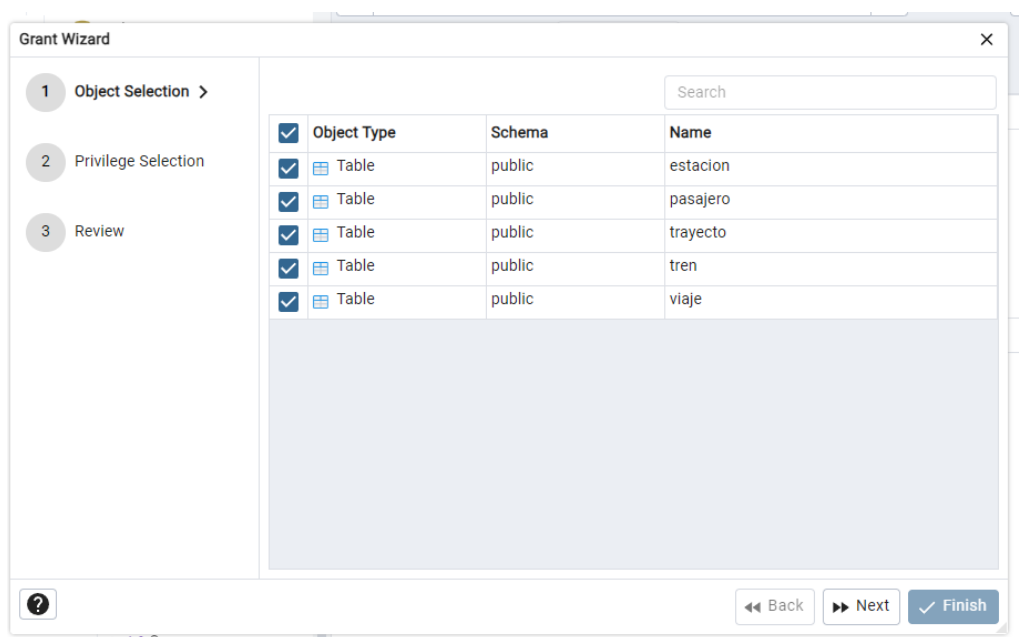
Para eliminar el usuario se hará de la siguiente forma:

```
transporte=# DROP ROLE usuario_consulta;
DROP ROLE
transporte=# \dg
```

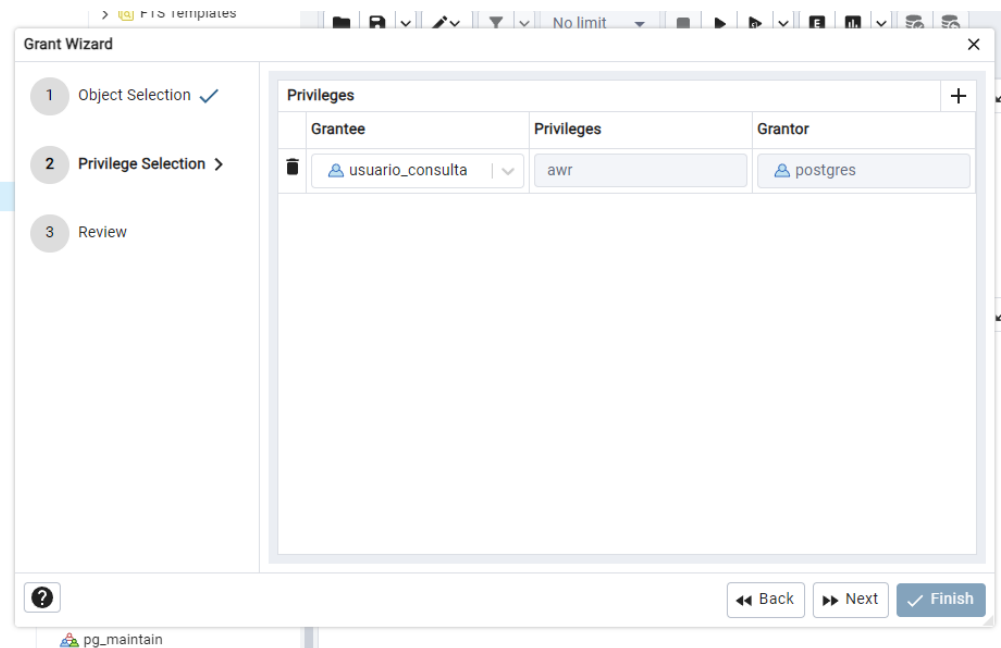
Nombre de rol	Lista de roles	Atributos
postgres		Superusuario, Crear rol, Crear BD, Replicación, Ignora RLS

```
transporte=#
```

Para asignar los permisos a una tabla de un usuario en concreto, nos iremos a la siguiente parte:



Este usuario solo permitirá seleccionar, actualizar y crear



Si comprobamos la configuración de la tabla, vemos que se han asignado correctamente los permisos:

```

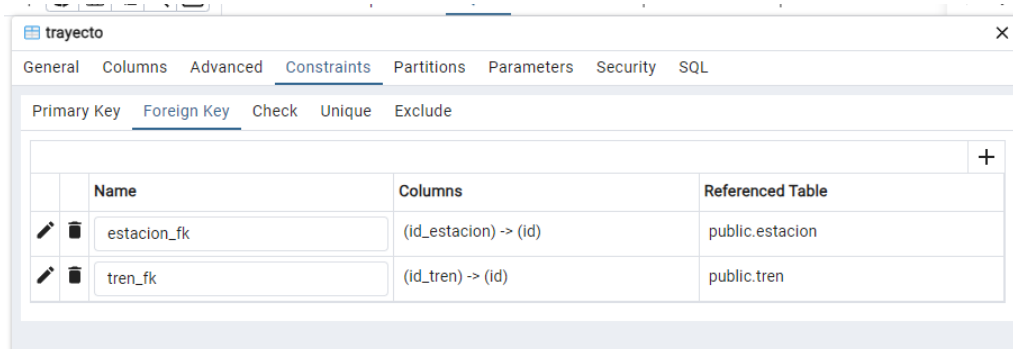
1  -- Table: public.estacion
2
3  -- DROP TABLE IF EXISTS public.estacion;
4
5  CREATE TABLE IF NOT EXISTS public.estacion
6  (
7      id integer NOT NULL DEFAULT nextval('estacion_id_seq'::regclass),
8      nombre character varying COLLATE pg_catalog."default" NOT NULL,
9      direccion character varying COLLATE pg_catalog."default" NOT NULL,
10     CONSTRAINT estacion_pk PRIMARY KEY (id)
11 )
12
13 TABLESPACE pg_default;
14
15 ALTER TABLE IF EXISTS public.estacion
16     OWNER to postgres;
17
18 REVOKE ALL ON TABLE public.estacion FROM usuario_consulta;
19
20 GRANT ALL ON TABLE public.estacion TO postgres;
21
22 GRANT INSERT, UPDATE, SELECT ON TABLE public.estacion TO usuario_consulta;

```

Llaves foráneas

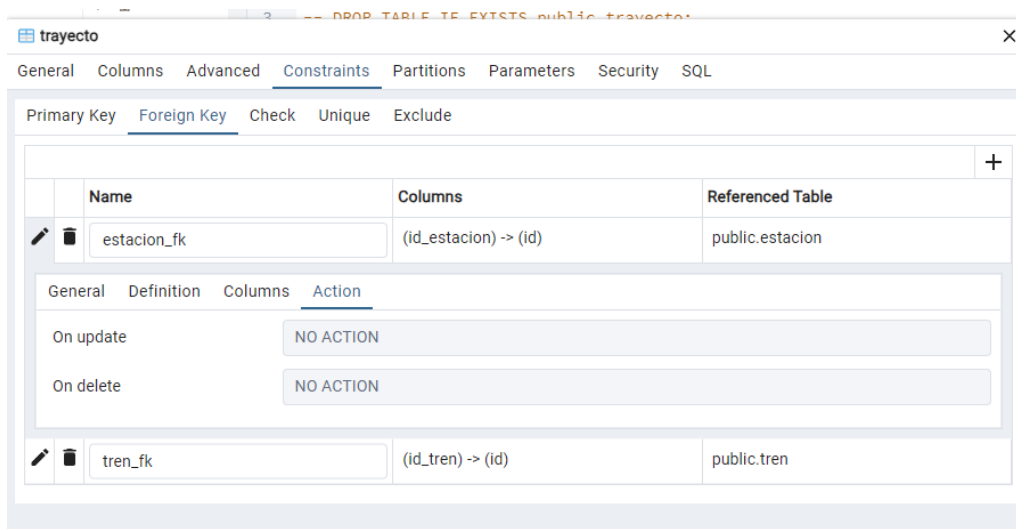
Llaves foráneas se refiere a la relación que tienen las tablas. En el apartado de ACIS, es la consistencia.

En el caso anterior, estableceremos las relaciones con las otras tablas:



La pestaña de acción, es importante, debido a que si hay un cambio, le decimos a la base de datos, que tenemos que hacer:

- NO ACTION: No hacer nada
- RESTRICT: Decirle a Postgres que no podemos permitir que la tabla cambie algo.
- CASCADE: Si cambio la tabla de origen, la tabla destino tambien cambia.
- SET NULL quiere decir que nuestra columna en esa fila va a dejar de tener por ejemplo el ID que tenia asociado un 77 y va a convertirse en NULL. Esto por que la tabla destino recibe un cambio y le decimos aPostgres que lo ponga en nulo.
- SET DEFAULT: Si hay un cambio en la tabla origen nuestra tabla destino ponga un valor predeterminado. En un ejemplo un id podra quedar NULL.



También lo podemos hacer mediante código:

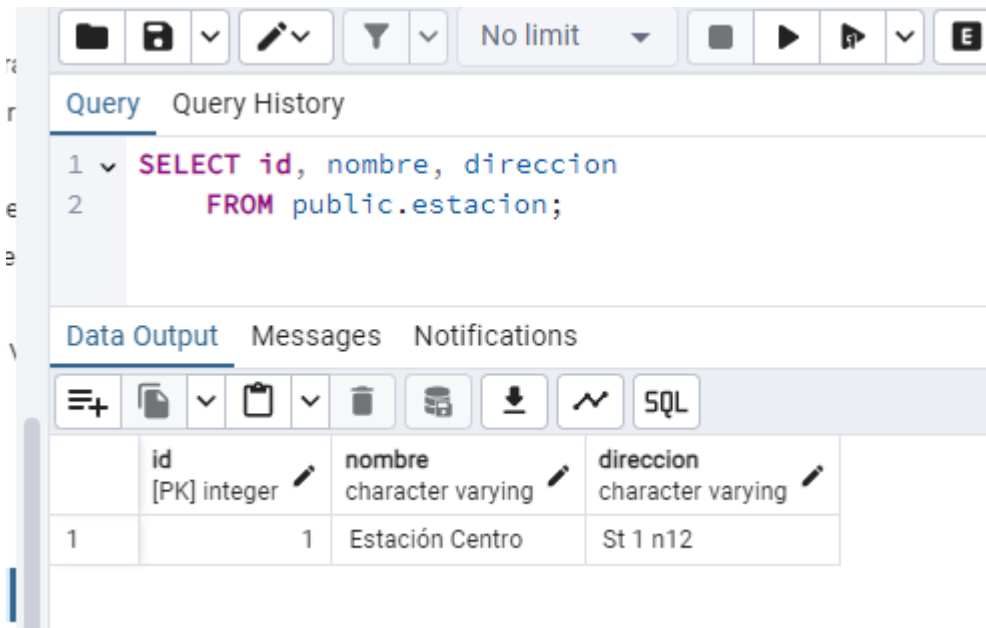
```
Query  Query History
1  -- Agregamos llave foranea de id_estacion a la tabla trayecto
2  ALTER TABLE public.trayecto
3      ADD CONSTRAINT trayecto_estacion_fkey FOREIGN KEY (id_estacion)
4      REFERENCES public.estacion (id) MATCH SIMPLE
5      ON UPDATE CASCADE
6      ON DELETE CASCADE
7      NOT VALID;
8
9  -- Agregamos llave foranea de id_tren a la tabla trayecto
10 ALTER TABLE public.trayecto
11     ADD CONSTRAINT trayecto_tren_fkey FOREIGN KEY (id_tren)
12     REFERENCES public.tren (id) MATCH SIMPLE
13     ON UPDATE CASCADE
14     ON DELETE CASCADE
15     NOT VALID;
16
17 -- Agregamos llave foranea de id_trayecto a la tabla viaje
18 ALTER TABLE public.viaje
19     ADD CONSTRAINT viaje_trayecto_fkey FOREIGN KEY (id_trayecto)
20     REFERENCES public.trayecto (id) MATCH SIMPLE
21     ON UPDATE CASCADE
22     ON DELETE CASCADE
23     NOT VALID;
24
25 -- Agregamos llave foranea de id_pasajero a la tabla viaje
26 ALTER TABLE public.viaje
27     ADD CONSTRAINT viaje_pasajero_fkey FOREIGN KEY (id_pasajero)
28     REFERENCES public.pasajero (id) MATCH SIMPLE
29     ON UPDATE CASCADE
30     ON DELETE CASCADE
31     NOT VALID;
32
33 -- Para borrar una llave foranea por si tienes algun error
34 ALTER TABLE public.viaje DROP CONSTRAINT viaje_trayecto_fkey;
```

Inserción y consulta de datos

Para crear datos se realiza de la siguiente forma:

```
Query  Query History
1  INSERT INTO public.estacion(
2      nombre, direccion)
3      VALUES ('Estación Centro', 'St 1');
```

Para consultar los datos, se utiliza la sentencia SELECT:

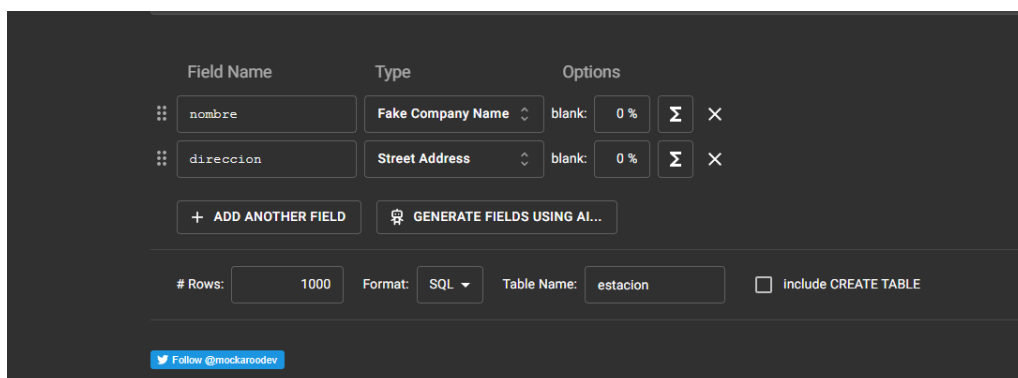


Para eliminar un registro se realizará de la siguiente forma:



Insertar datos de forma masiva

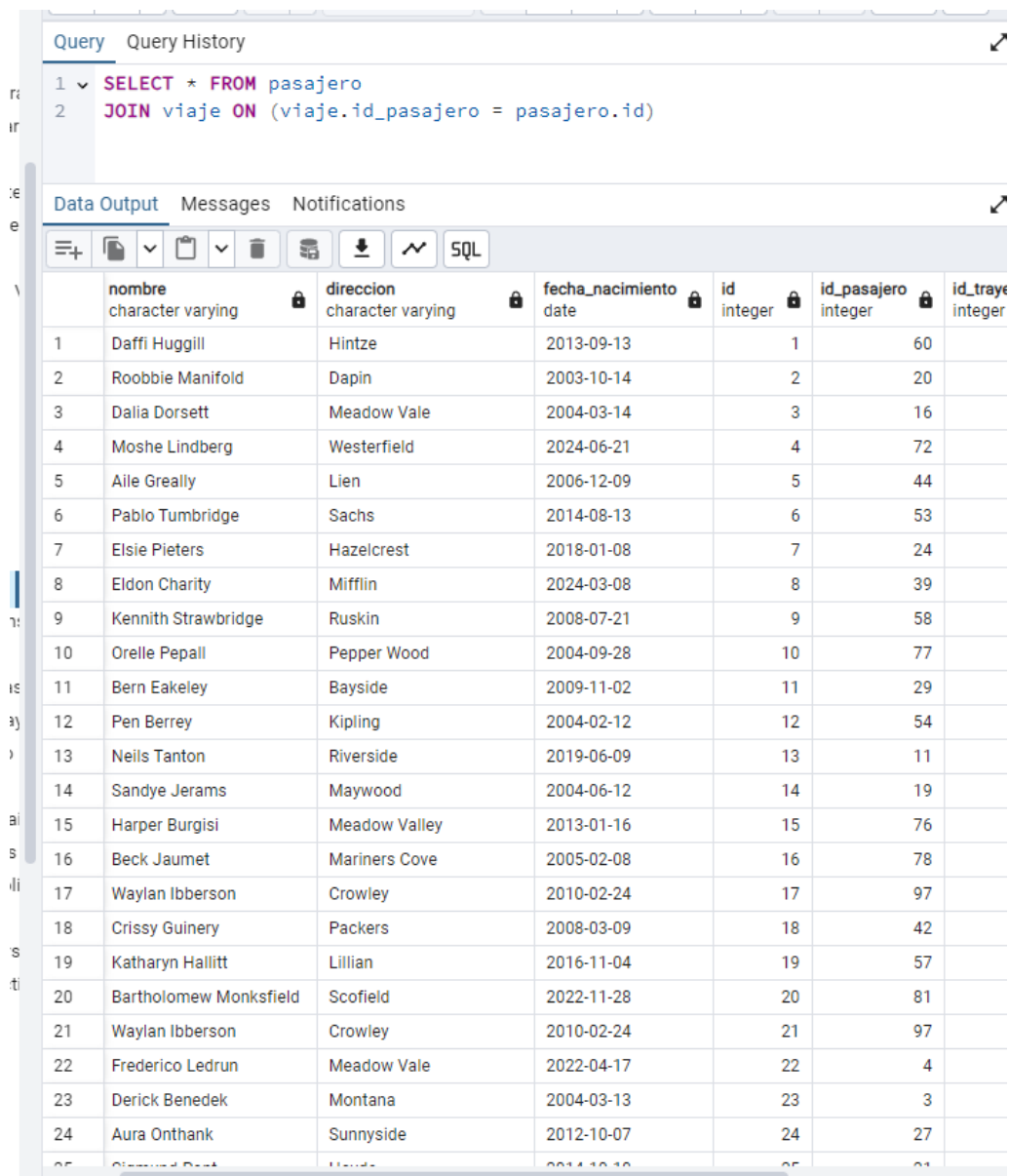
Para poder insertar datos de una forma masiva, se utilizará la herramienta Mockaroo



Simplemente hay que añadir los campos de las tabla, el tipo de dato que se quiera poner de forma aleatoria, y poner el nombre de la tabla.

Consultar tablas

Para unir varias tablas se realizaría de la siguiente forma:



The screenshot shows a database query tool interface. The 'Query' tab is active, displaying a SQL query that joins the 'pasajero' and 'viaje' tables. The 'Data Output' tab is also visible, showing the results of the query in a table format. The table has columns for passenger details and travel information.

	nombre character varying	direccion character varying	fecha_nacimiento date	id integer	id_pasajero integer	id_traye integer
1	Daffi Huggill	Hintze	2013-09-13	1	60	
2	Roobbie Manifold	Dapin	2003-10-14	2	20	
3	Dalia Dorsett	Meadow Vale	2004-03-14	3	16	
4	Moshe Lindberg	Westerfield	2024-06-21	4	72	
5	Aile Greally	Lien	2006-12-09	5	44	
6	Pablo Tumbridge	Sachs	2014-08-13	6	53	
7	Elsie Pieters	Hazelcrest	2018-01-08	7	24	
8	Eldon Charity	Mifflin	2024-03-08	8	39	
9	Kennith Strawbridge	Ruskin	2008-07-21	9	58	
10	Orelle Pepall	Pepper Wood	2004-09-28	10	77	
11	Bern Eakeley	Bayside	2009-11-02	11	29	
12	Pen Berrey	Kipling	2004-02-12	12	54	
13	Neils Tanton	Riverside	2019-06-09	13	11	
14	Sandye Jerams	Maywood	2004-06-12	14	19	
15	Harper Burgisi	Meadow Valley	2013-01-16	15	76	
16	Beck Jaumet	Mariners Cove	2005-02-08	16	78	
17	Waylan Ibberson	Crowley	2010-02-24	17	97	
18	Crissy Guinery	Packers	2008-03-09	18	42	
19	Katharyn Hallitt	Lillian	2016-11-04	19	57	
20	Bartholomew Monksfield	Scofield	2022-11-28	20	81	
21	Waylan Ibberson	Crowley	2010-02-24	21	97	
22	Frederico Ledrun	Meadow Vale	2022-04-17	22	4	
23	Derick Benedek	Montana	2004-03-13	23	3	
24	Aura Onthank	Sunnyside	2012-10-07	24	27	
25	Stanford Bort	Montana	2014-10-10	25	21	

Para ver que pasajeros no han hecho ningún viaje, se haría de la siguiente forma:

transporte/postgres@PostgreSQL 17* x

transporte/postgres@PostgreSQL 17

Query Query History

```

1 SELECT * FROM pasajero
2 LEFT JOIN viaje ON (viaje.id_pasajero = pasajero.id)
3 WHERE viaje.id is NULL

```

Data Output Messages Notifications

SQL

	id integer	nombre character varying	direccion character varying	fecha_nacimiento date	id integer	id_pasajero integer
1	101	Corey Portt	Lake View	2005-01-05	[null]	[null]
2	25	Shirlene Chittie	Heath	2005-01-23	[null]	[null]
3	26	Werner Monini	Dawn	2001-02-19	[null]	[null]
4	93	Jammie Coakley	Mayer	2018-05-05	[null]	[null]
5	89	Farah Leward	Dayton	2012-09-15	[null]	[null]
6	33	Godart Tant	Anniversary	2012-11-24	[null]	[null]
7	34	Israel Dewerson	Waywood	2014-09-06	[null]	[null]
8	18	Maxy Baudic	Meadow Valley	2013-09-18	[null]	[null]
9	15	Anestassia Meadowcroft	Arizona	2011-03-31	[null]	[null]
10	56	Arnud Borham	Nobel	2008-08-23	[null]	[null]
11	40	Colby Grennan	Basil	2024-07-03	[null]	[null]
12	13	Henryetta Bodsworth	Sycamore	2023-10-04	[null]	[null]
13	91	Andrew Kenward	Ronald Regan	2008-04-11	[null]	[null]
14	65	Astrid Kinworthy	Logan	2012-02-22	[null]	[null]
15	37	Tirrell Zamorrano	Bonner	2024-08-18	[null]	[null]
16	85	Janette Copp	Meadow Ridge	2024-01-16	[null]	[null]
17	32	Burty Eyton	Claremont	2024-03-27	[null]	[null]
18	100	Charlene O'Henecan	Ridgeway	2021-03-21	[null]	[null]
19	38	Everett Bromige	Saint Paul	2012-02-10	[null]	[null]
20	8	Artur Elwyn	Lakewood Gardens	2001-02-10	[null]	[null]
21	80	Hi Dunkerton	Heffernan	2022-07-18	[null]	[null]
22	99	Hagen Chatburn	Swallow	2012-04-23	[null]	[null]
23	48	Reilly Aveson	Larry	2013-02-28	[null]	[null]
24	28	Tricia Marikhin	Hagan	2010-12-13	[null]	[null]
25	24	Rebecca Boudala	Riverside	2001-04-04	[null]	[null]

Funciones especiales

- ON CONFLICT DO
- RETURNING
- LIKE / ILIKE
- IS / IS NOT

ON CONFLICT DO

Esta instruccion 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, actualize los datos.

```
INSERT INTO pasajero (id, nombre, direccion_residencia, fecha_nacimiento)
  values (1, '', '', '2010-01-01')
ON CONFLICT (id) DO
  UPDATE SET
    nombre = '', direccion_residencia='', fecha_nacimiento
    ='2010-01-01';
```

RETURNING

Returning nos devuelve una consulta select de los campos sobre los que ha tenido efecto la instruccion.

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

```
INSERT INTO tren (modelo, capacidad)
  VALUES('modelo x', 100)
  RETURNING id;

/*
Opcionalmente tambien puedes solicitar todos los campos o a
lguno de ellos
*/

INSERT INTO tren (modelo, capacidad)
  VALUES('modelo x', 100)
  RETURNING id;

INSERT INTO tren (modelo, capacidad)
  VALUES('modelo x', 100)
  RETURNING id, capacidad;
```

Like / Ilike

Las funciones like y ilike sirven para crear consultas a base de expresiones regulares.

Like considera mayusculas y minusculas, mientras que ilike solo considera las letras.

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

```
-- Usando LIKE
SELECT * FROM PASAJERO
WHERE pasajero.nombre LIKE '%O'
-- No devuelve nada, porque ningun nombre terminara con una
letra mayuscula

-- Usando ILIKE
SELECT * FROM PASAJERO
WHERE pasajero.nombre ILIKE '%O'
-- Devolvera los nombres que terminen con o, independiente
si es mayuscula o minuscula.
```

IS / IS NOT

Permite hacer comprobacion de valores especiales como **null**

Ejemplo: Consultemos a todos los usuarios que tengan como direccion_residencia NULL

```
-- IS
SELECT * FROM PASAJERO
WHERE pasajero.nombre IS null;
```

Ahora a los que si tengan la direccion_recidencia con algun valor

```
-- IS NOT
SELECT * FROM PASAJERO
WHERE pasajero.nombre IS NOT null;
```

Funciones especiales avanzadas

- COALESCE
- NULLIF
- GREATEST
- LEAST
- BLOQUES ANONIMOS

```
-- Si el nombre es null, entonces me muestra 'No Aplica'
SELECT id, COALESCE(nombre, 'No Aplica') AS nombre, direccion_residencia FROM public.pasajero
WHERE id = 1;
```

```
-- Si dos campos son iguales ---> 0: False
SELECT NULLIF(0,1);
```

```
-- Devuelve el mayor
SELECT GREATEST(0,1,2,5,2,3,6,10,2,1,20);
```

```
-- Devuelve en menor
SELECT LEAST(0,1,2,5,2,3,6,10,2,1,20);
```

```
-- Devuelve si es Niño o Mayor
SELECT id, nombre, direccion_residencia, fecha_nacimiento,
       (CASE
        WHEN (fecha_nacimiento) > '2015-01-01' THEN
            'Niño'
        ELSE
            'Mayor'
        END)
FROM public.pasajero;
```

```
-- Reto
SELECT id, nombre, direccion_residencia, fecha_nacimiento,
       (CASE
        WHEN (CURRENT_DATE - fecha_nacimiento)/365 >= 18 THEN
            '>Mayor'
        ELSE
```

```
'&lt;Menor '  
    END)  
FROM public.pasajero;
```

Vista

Una vista es una alternativa para mostrar datos de varias tablas. Una vista es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos como un objeto. . Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista. .

Las vistas permiten:

- **Ocultar información:** generando el acceso a algunos datos y manteniendo oculto el resto de la información que no se incluye en la vista. El usuario solo puede consultar la vista. .
- **Simplificar la administración de los permisos de usuario:** se pueden dar al usuario permisos para que solamente pueda acceder a los datos a través de vistas, en lugar de concederle permisos para acceder a ciertos campos, así se protegen las tablas base de cambios en su estructura. .
- **Mejorar el rendimiento:** se puede evitar tipear instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas. . Podemos crear vistas con: un subconjunto de registros y campos de una tabla; una unión de varias tablas; una combinación de varias tablas; un resumen estadístico de una tabla; un subconjunto de otra vista, combinación de vistas y tablas. .

La sintaxis básica parcial para crear una vista es la siguiente:

```
create view NOMBREVISTA as  
    SENTENCIAS SELECT  
    from TABLA;  
El contenido de una vista se muestra con un "select":
```

```
select *from NOMBREVISTA;
```

- Vista Volatil
- Vista Materializada: persistente (Ayer)

```
-- Creamos la vista
CREATE OR REPLACE VIEW public.rango_view
AS
    SELECT *,
        CASE
            WHEN fecha_nacimiento > '2015-01-01' THEN
                'Mayor'
            ELSE
                'Menor'
            END AS tipo
    FROM pasajero ORDER BY tipo;
ALTER TABLE public.rango_view OWNER TO postgres;

-- mostramos la vista
SELECT * FROM public.rango_view;

-- Vistas Materializada, no se cambia a menos que queramos
que se cambie
SELECT * FROM viaje WHERE inicio > '22:00:00';

CREATE MATERIALIZED VIEW public.despues_noche_mview
AS
    SELECT * FROM viaje WHERE inicio > '22:00:00';
WITH NO DATA;
ALTER TABLE public.despues_noche_mview OWNER TO postgres;

-- observamos la vista
SELECT * FROM despues_noche_mview;

-- Damos refresh
REFRESH MATERIALIZED VIEW despues_noche_mview;
```



```
-- Borramos una tupla de viaje cuando el id = 2, para obser  
var que no se borro  
DELETE FROM viaje WHERE id = 2;
```

Otro concepto importante son las tablas temporales que se guardan en disco y se eliminan al cerrar la sesión:

```
create temporary table viajes_temp AS  
select * from viajes;
```