

## 5.1 Tutorial Bases de datos columnares (video relacionado: 5.6 Distribution styles. Contenido relacionado: Tipos de Distribution Styles en Amazon redshift).

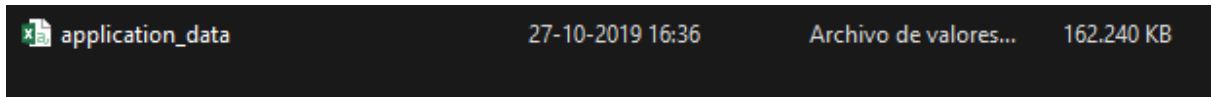
Actualmente trabajas en un banco popular de la ciudad, en tu primer día de trabajo te reunes con el Data Manager, quien te pide que analices qué tan ventajoso puede ser utilizar una base de datos columnar como amazon Redshift en comparación con una base de datos orientada a filas como PostgreSQL a la hora de hacer consultas de tipo analítica teniendo en cuenta la siguiente [base de datos](#) y las siguientes restricciones:

1. La descripción de las columnas se encuentra [aquí](#)
2. Nuestras variables de interés son:
  - a. TARGET,"Target variable (1 – client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 – all other cases)"
  - b. NAME\_CONTRACT\_TYPE,Identification if loan is cash or revolving
  - c. CODE\_GENDER,Gender of the client,
  - d. AMT\_CREDIT,Credit amount of the loan
3. El objetivo será obtener agrupaciones del AMT:CREDIT de acuerdo con tipo de contrato, género y TARGET.

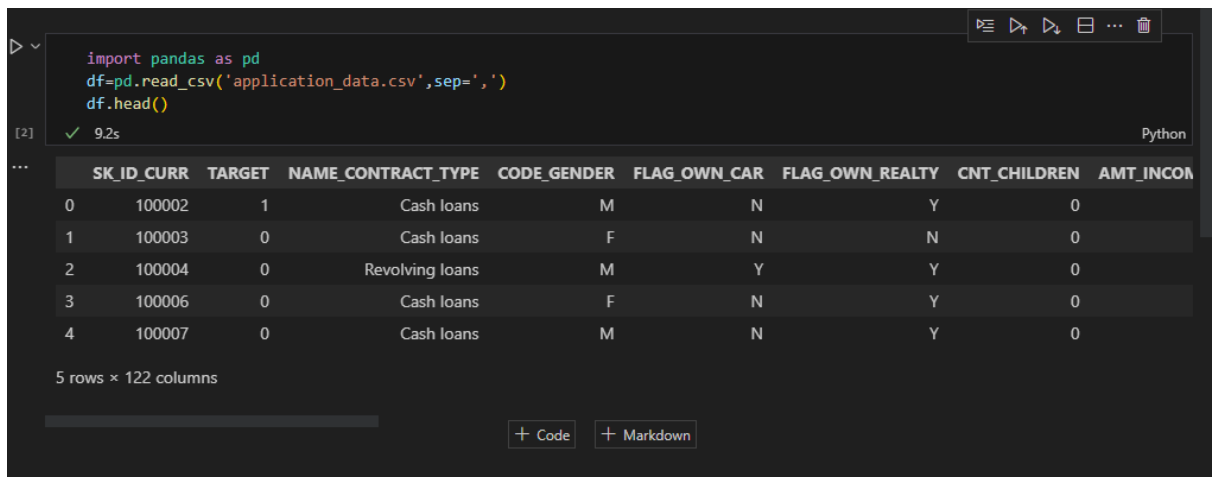
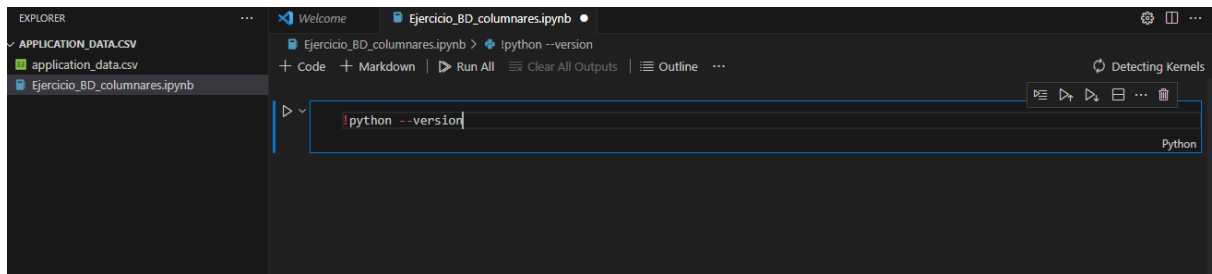
**En la próxima página podrás ver la solución.**

## ❖ Solución:

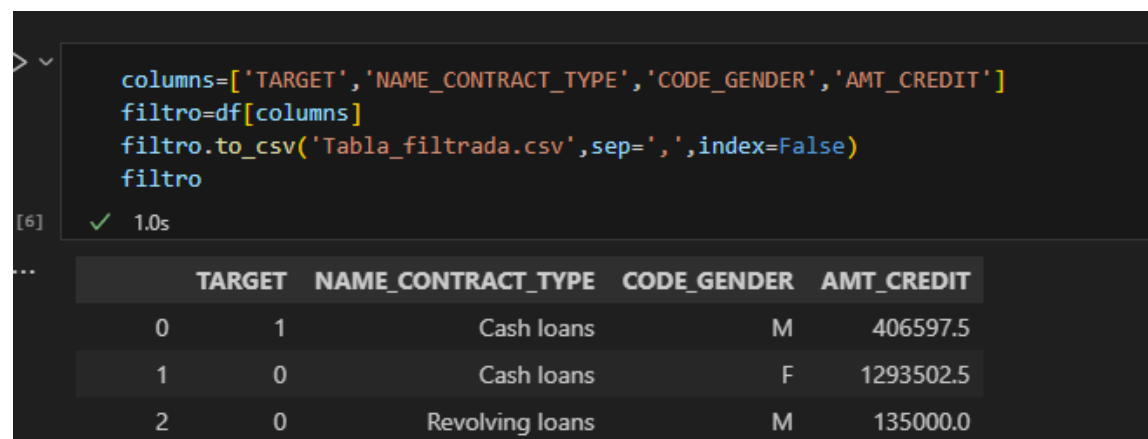
1. Descargamos el dataset y lo descomprimos:



2. Luego abrimos nuestro editor de código (e.g Visual studio) y leemos nuestro archivo con pandas:



3. Hacemos el subset de las variables de interés y exportamos esta tabla como csv:

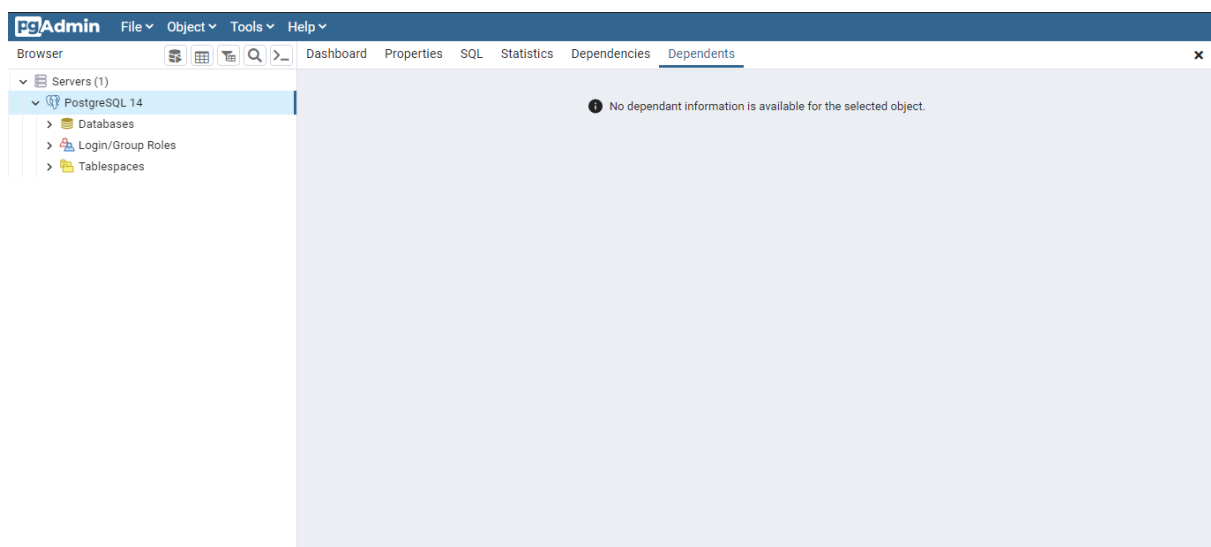


**NOTA: TE RECOMIENDO USAR SOLO 5000 filas del dataset para que no demores demasiado importando los datos en Redshift.**

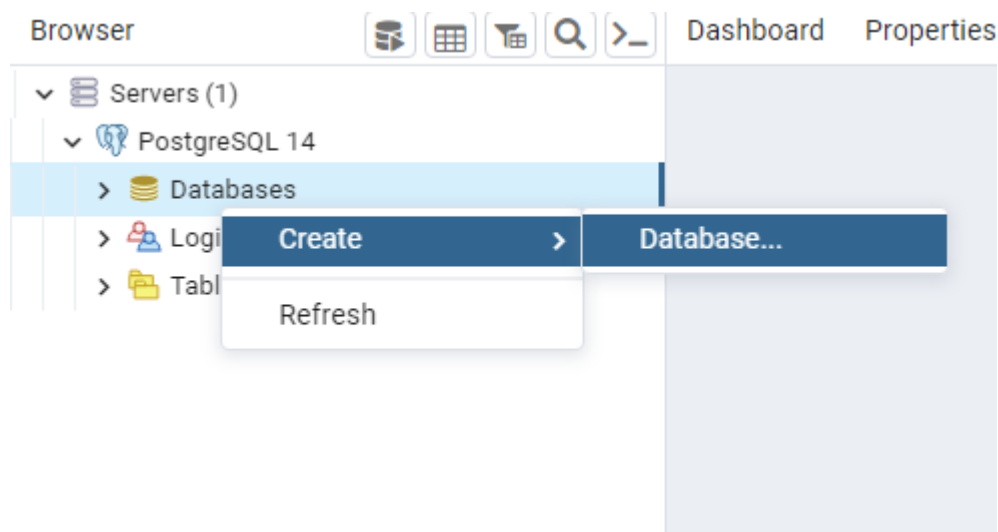
4. Ahora podemos abrir PgAdmin:



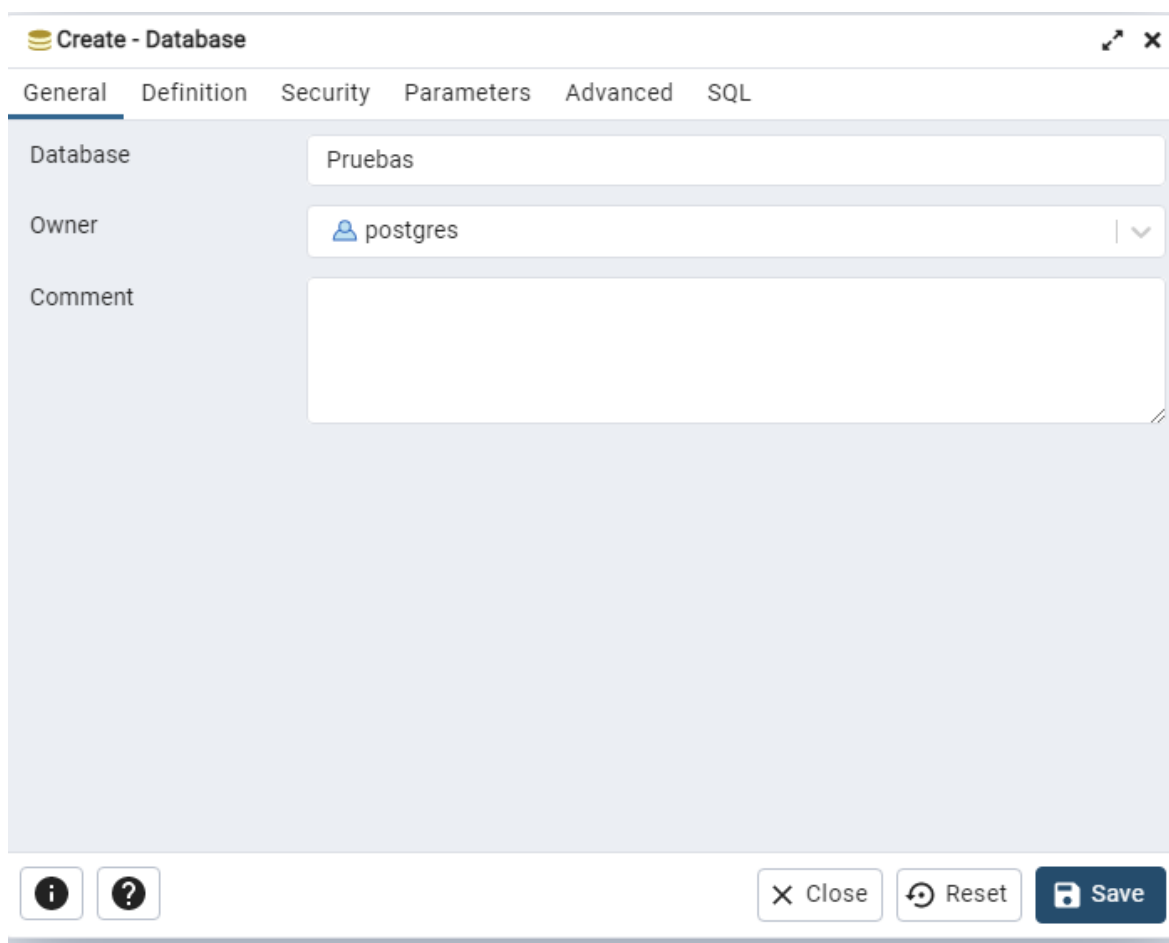
5. Dentro de pgadmin podemos crear una base de datos o utilizar alguna ya existen:



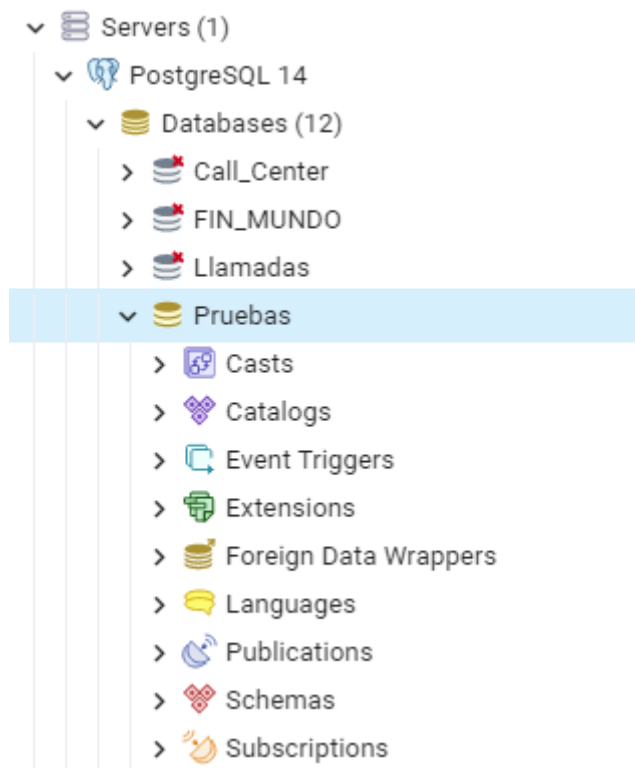
6. Creamos una base de datos llamada Pruebas dando click derecho en Databases > Crear Database:



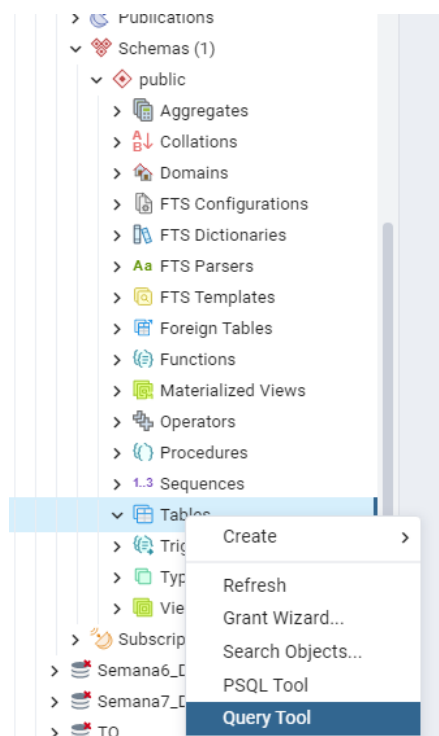
7. Podemos dejar todo por default asignamos el nombre Pruebas y damos en Save:



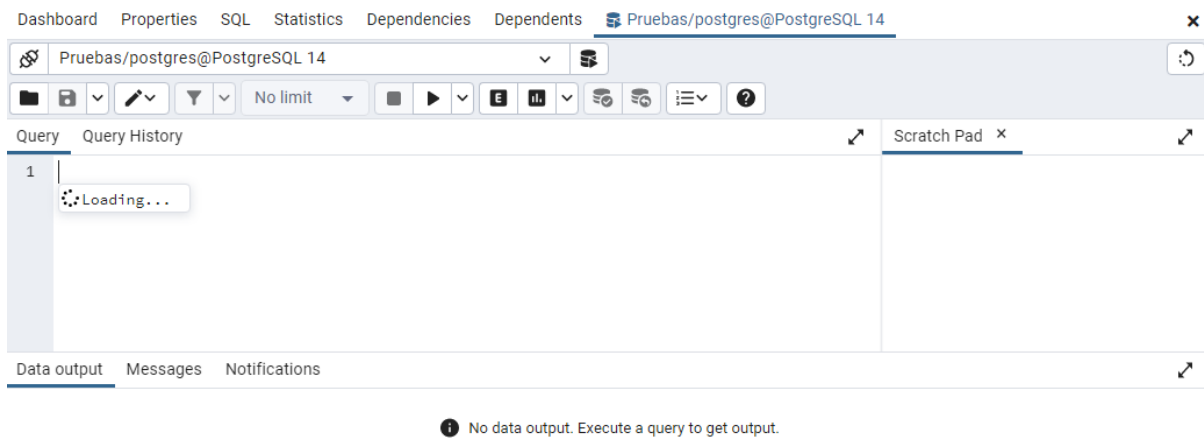
8. Ahora debería aparecer nuestra base de datos:



9. Vamos donde dice esquemas > Tablas > Click derecho y luego seleccionamos Query Tool:



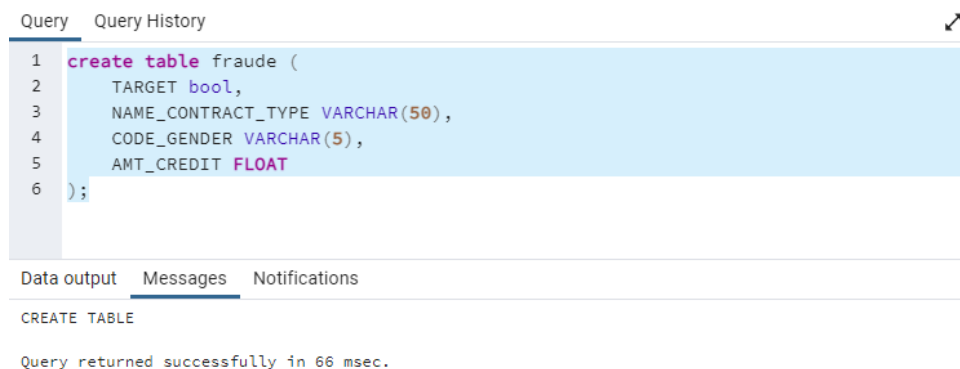
10. Aparecerá una ventana donde podremos crear nuestra tabla:



11. Creamos nuestra tabla con el siguiente comando de sql:

```
```sql
create table fraude (
    TARGET bool,
    NAME_CONTRACT_TYPE VARCHAR(50),
    CODE_GENDER VARCHAR(5),
    AMT_CREDIT FLOAT
);
```
```

Así entonces deberíamos tener nuestra tabla creada:



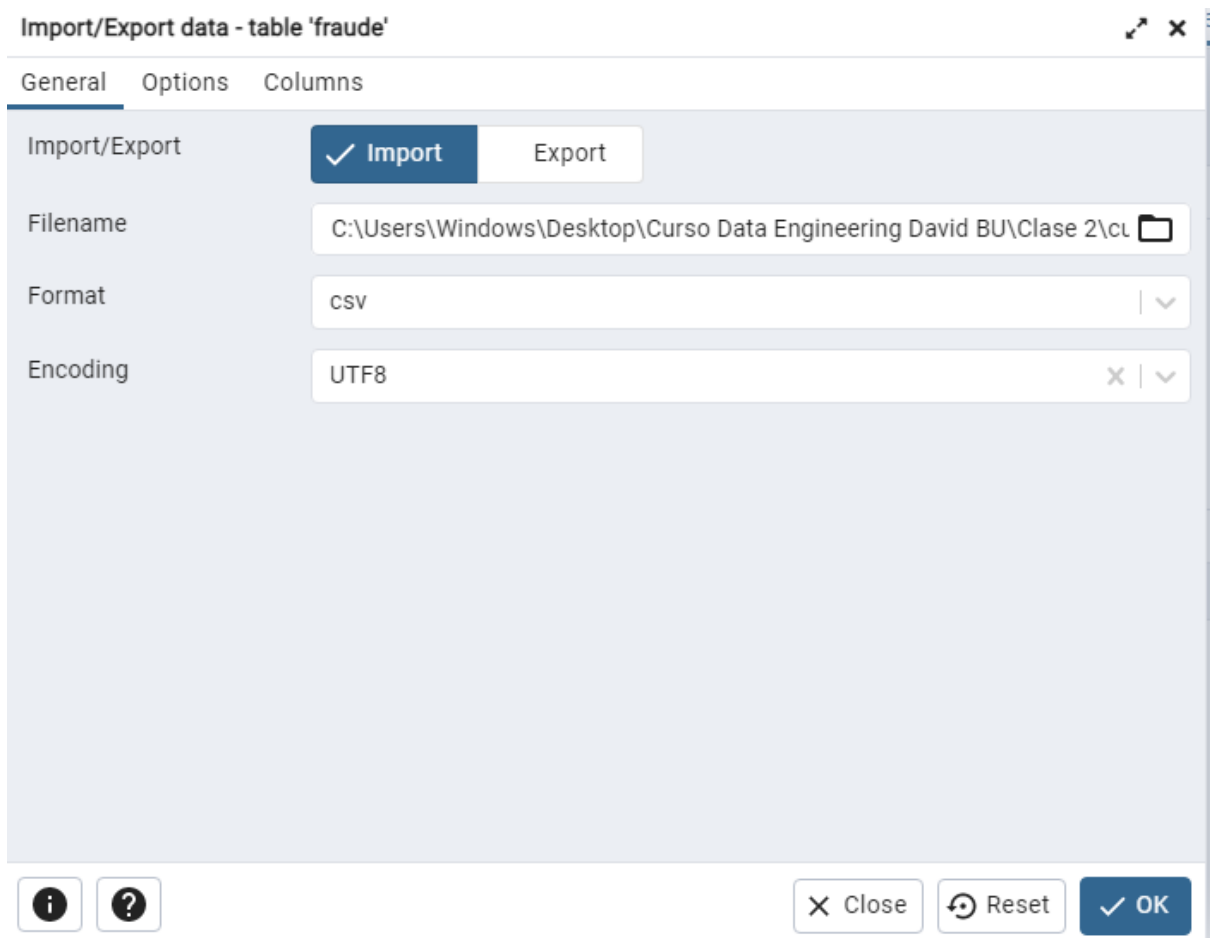
12. Ahora podemos ver nuestra tabla:

The screenshot shows a database query interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a single query: `1 SELECT * FROM fraude`. Below the query editor, there are tabs for 'Data output', 'Messages', and 'Notifications'. The 'Data output' tab is active, showing a table with four columns: `target` (boolean), `name_contract_type` (character varying (50)), `code_gender` (character varying (5)), and `amt_credit` (double precision). Each column name is followed by a lock icon.

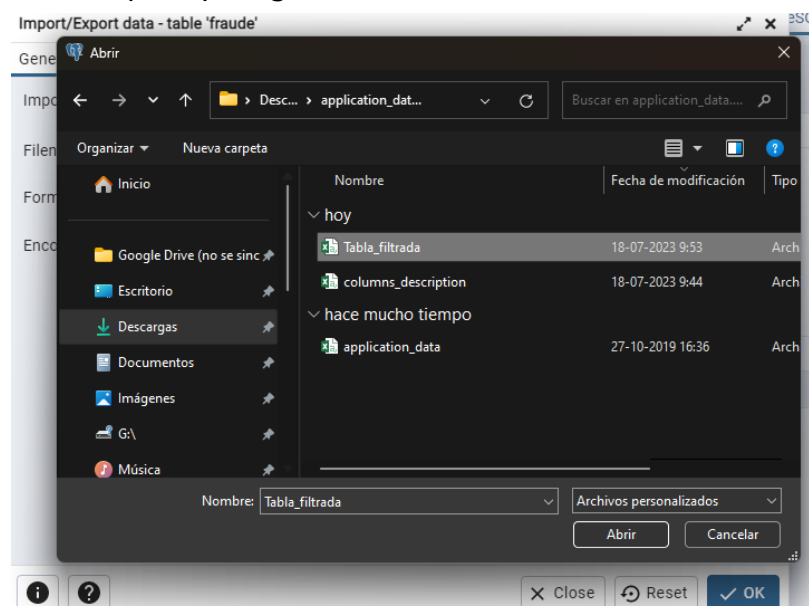
13. Procedemos a llenar la tabla con registros. Click derecho en el nombre de la tabla > Import/Export:

The screenshot shows a database browser interface. A right-click context menu is open over a table named 'fraude'. The menu options include: 'Count Rows', 'Create', 'Delete/Drop', 'Refresh...', 'Restore...', 'Backup...', 'Drop Cascade', 'Import/Export Data...', 'Reset Statistics', 'Maintenance...', 'Scripts', 'Truncate', 'View/Edit Data', 'Search Objects...', 'PSQL Tool', 'Query Tool', and 'Properties...'. The 'Import/Export Data...' option is highlighted.

14. Aparecerá una ventana como esta:

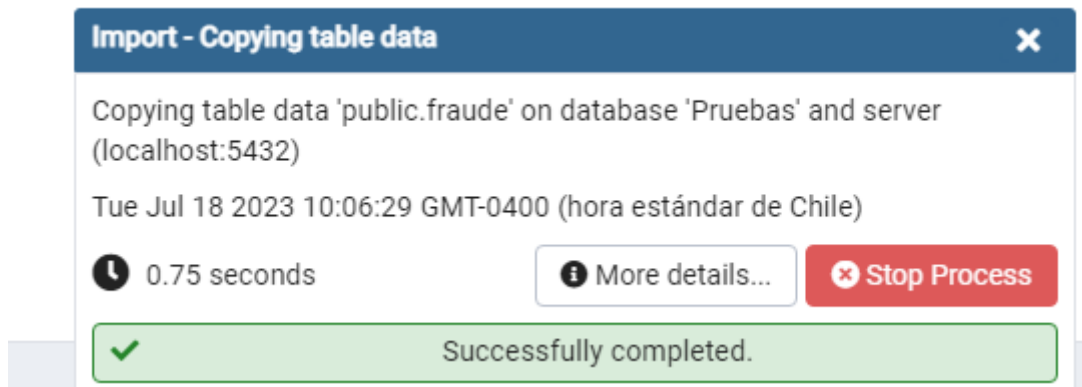


15. Seleccionamos import y elegimos nuestro archivo:





16. Damos en OK y deberá salir un mensaje como este:



17. Verificamos nuestra tabla:

Query Query History

```
1 SELECT * FROM fraude
```

Scratch Pad

Data output Messages Notifications

|   | target<br>boolean | name_contract_type<br>character varying (50) | code_gender<br>character varying (5) | amt_credit<br>double precision |
|---|-------------------|--|--------------------------------------|--------------------------------|
| 1 | true              | Cash loans                                   | M                                    | 406597.5                       |
| 2 | false             | Cash loans                                   | F                                    | 1293502.5                      |
| 3 | false             | Revolving loans                              | M                                    | 135000                         |
| 4 | false             | Cash loans                                   | F                                    | 312682.5                       |
| 5 | false             | Cash loans                                   | M                                    | 513000                         |
| 6 | false             | Cash loans                                   | M                                    | 490495.5                       |
| 7 | false             | Cash loans                                   | F                                    | 1560726                        |
| 8 | false             | Cash loans                                   | M                                    | 1530000                        |
| 9 | false             | Cash loans                                   | F                                    | 1019610                        |

Total rows: 1000 of 307511 Query complete 00:00:00.271

Successfully run. Total query runtime: 271 msec. 307511 rows affected.

18. Ahora realizamos diversas columnas utilizando funciones de agregación:

```
```sql
SELECT name_contract_type,AVG(amt_credit)
FROM fraude
GROUP BY name_contract_type
```
```

Pruebas/postgres@PostgreSQL 14

Query
Query History

```

1 SELECT name_contract_type,AVG(amt_credit)
2 FROM fraude
3 GROUP BY name_contract_type

```

Data output
Messages
Notifications

|   | name_contract_type<br>character varying (50) | avg<br>double precision |
|---|--|-------------------------|
| 1 | Cash loans                                   | 627965.732537953        |
| 2 | Revolving loans                              | 324017.982171522        |

Total rows: 2 of 2

Query complete 00:00:06.916

Notarás que se demora cerca de 6.9 segundos, ten este tiempo presente para las comparaciones:

```

```sql
SELECT code_gender,AVG(amt_credit)
FROM fraude
GROUP BY code_gender
```

```

Query

Query History

1

SELECT code\_gender,AVG(amt\_credit)

2

FROM fraude

3

GROUP BY code\_gender

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

|   | code_gender<br>character varying (5) 🔒 | avg<br>double precision 🔒 |
|---|--|---------------------------|
| 1 | F                                      | 592766.7173051846         |
| 2 | M                                      | 611095.1970844953         |
| 3 | XNA                                    | 399375                    |

Total rows: 3 of 3

Query complete 00:00:03.815

En este otro caso demora cerca de 3.8 seg

**Ahora bien si hacemos algo más sofisticado como esto notarás que tarda cerca de ~20seg**

```

```sql
SELECT code_gender,name_contract_type,target,AVG(amt_credit)
FROM fraude
GROUP BY code_gender,name_contract_type,target
ORDER BY code_gender,name_contract_type,target
```

```

Query

Query History

1

SELECT code\_gender,name\_contract\_type,target,AVG(amt\_credit)

2

FROM fraude

3

GROUP BY code\_gender,name\_contract\_type,target

4

ORDER BY code\_gender,name\_contract\_type,target

Data output

Messages

Notifications

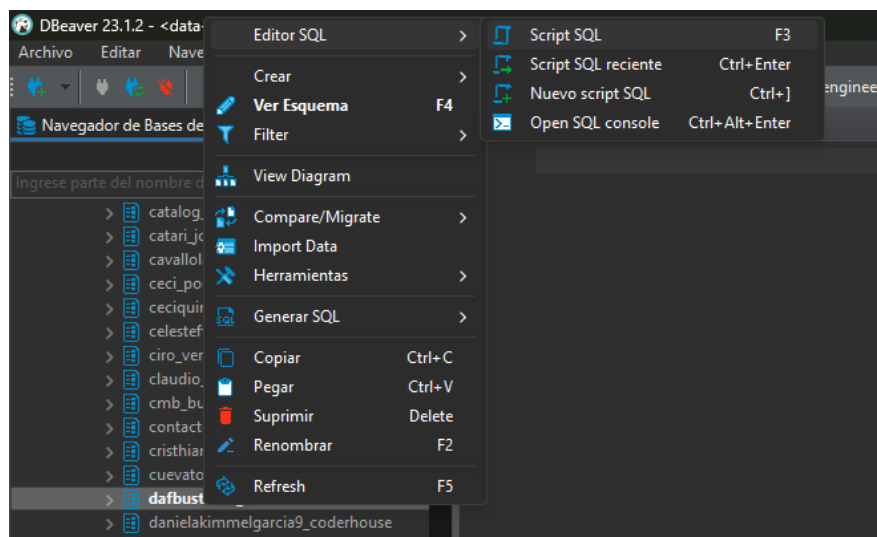
≡+

|   | code_gender<br>character varying (5) | name_contract_type<br>character varying (50) | target<br>boolean | avg<br>double precision |
|---|--------------------------------------|--|-------------------|-------------------------|
| 1 | F                                    | Cash loans                                   | false             | 626036.601097405        |
| 2 | F                                    | Cash loans                                   | true              | 585662.6165155786       |
| 3 | F                                    | Revolving loans                              | false             | 313441.144853534        |
| 4 | F                                    | Revolving loans                              | true              | 252504.79386385425      |
| 5 | M                                    | Cash loans                                   | false             | 645234.3800885889       |
| 6 | M                                    | Cash loans                                   | true              | 569412.4016742619       |
| 7 | M                                    | Revolving loans                              | false             | 357659.662475182        |
| 8 | M                                    | Revolving loans                              | true              | 263542.7807486631       |
| 9 | XNA                                  | Revolving loans                              | false             | 399375                  |

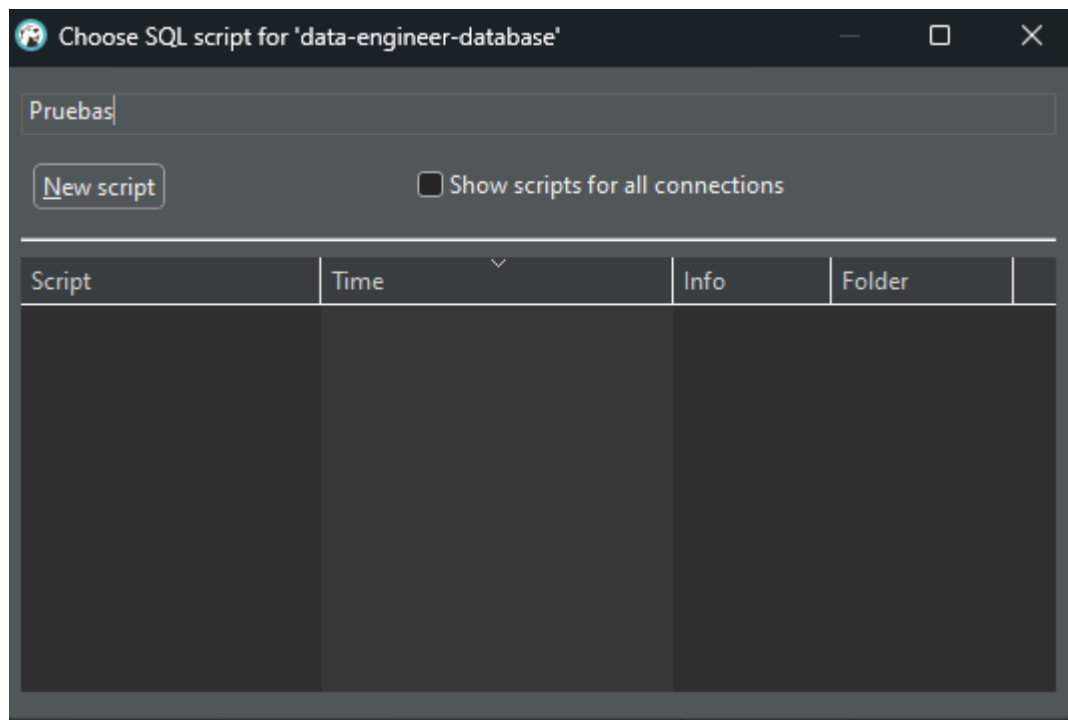
Total rows: 9 of 9

Query complete 00:00:17.710

19. Ahora hagamos lo mismo en Amazon Redshift. abrimos Dbeaver. elegimos nuestro nombre de esquema > Click derecho > Editor de SQL > New Script

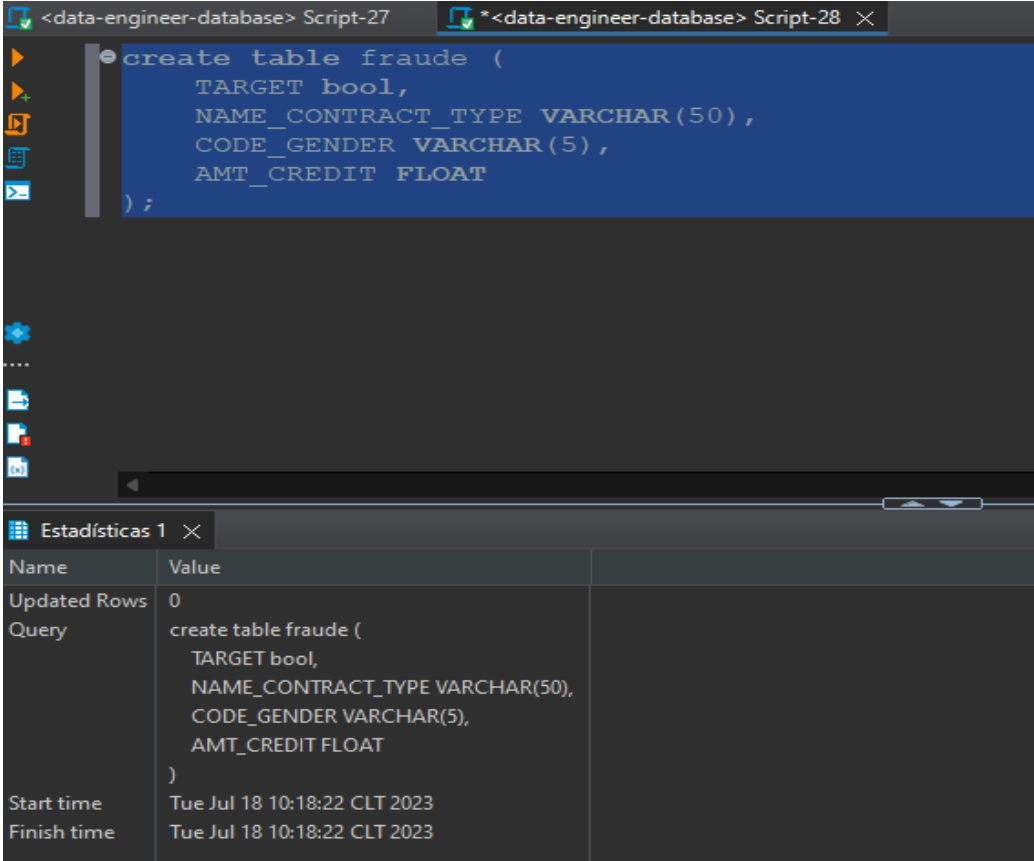


20. Ponemos un nombre a nuestro script y damos enter:



21. Creamos nuestra tabla:

```
```sql
create table fraude (
    TARGET bool,
    NAME_CONTRACT_TYPE VARCHAR(50),
    CODE_GENDER VARCHAR(5),
    AMT_CREDIT FLOAT
);
```



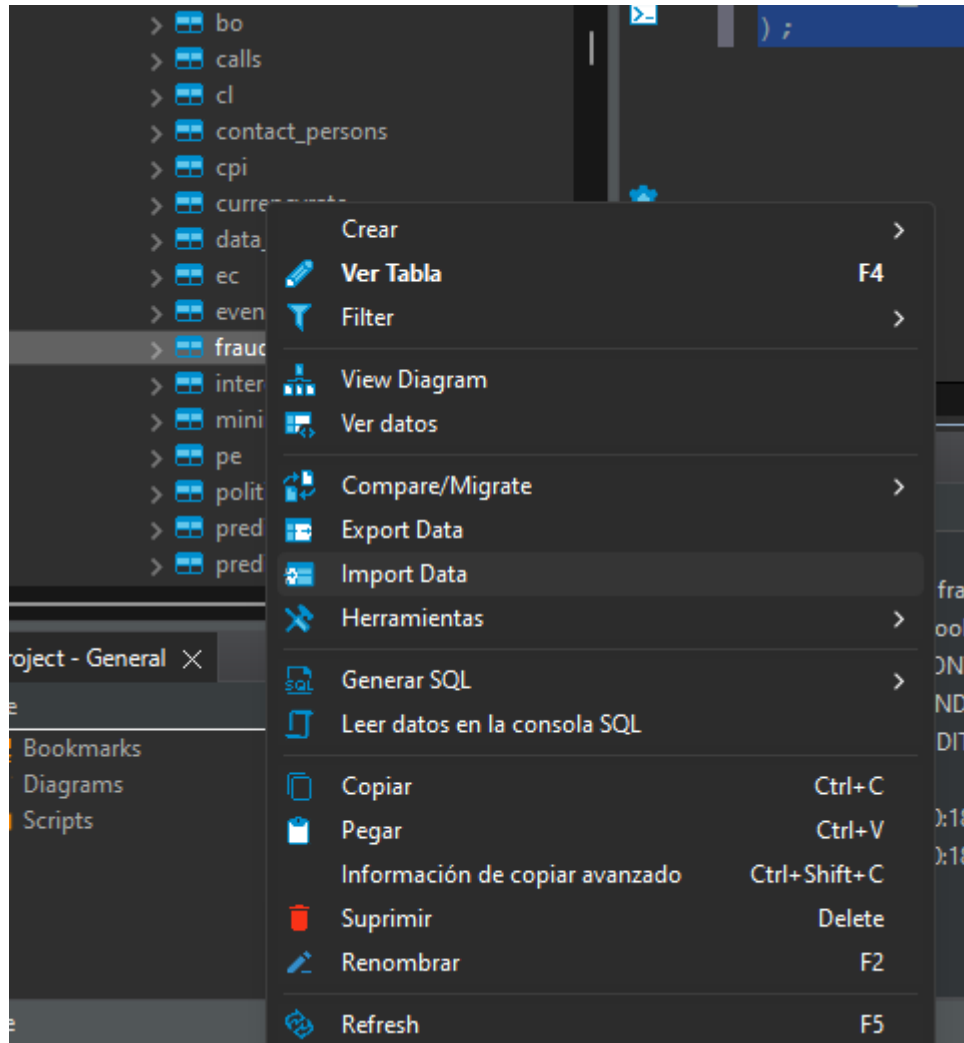
The screenshot shows a database IDE with two tabs: "<data-engineer-database> Script-27" and "<data-engineer-database> Script-28". The active tab, Script-28, contains the following SQL script:

```
create table fraude (
  TARGET bool,
  NAME_CONTRACT_TYPE VARCHAR(50),
  CODE_GENDER VARCHAR(5),
  AMT_CREDIT FLOAT
);
```

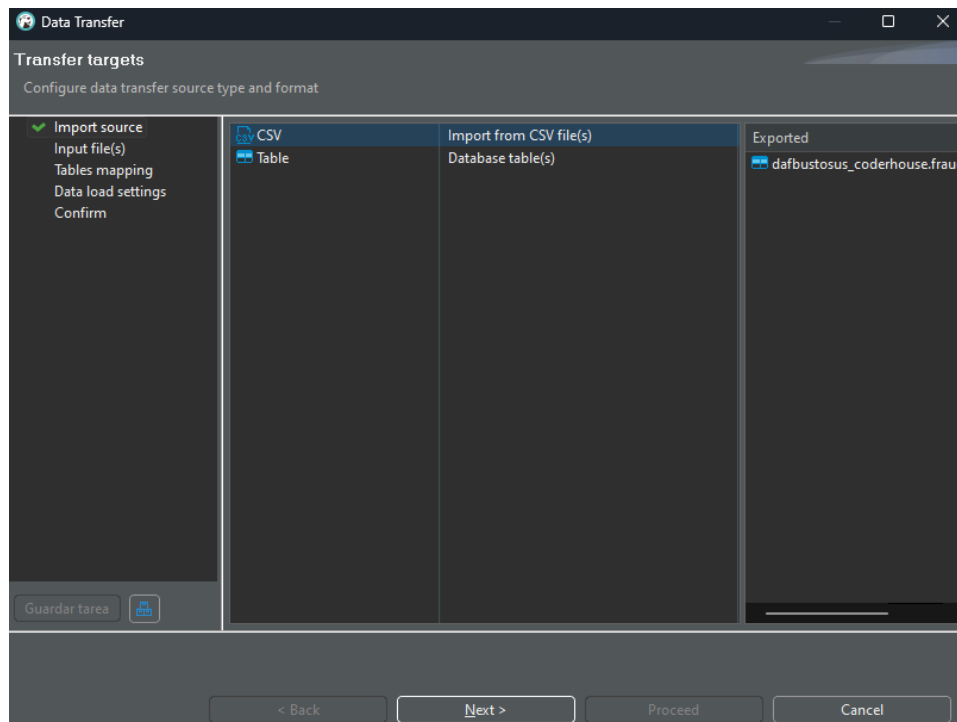
Below the script editor, there is a panel titled "Estadísticas 1" (Statistics 1) showing the execution results of the script. The panel has a table with the following data:

Name	Value
Updated Rows	0
Query	create table fraude (   TARGET bool,   NAME_CONTRACT_TYPE VARCHAR(50),   CODE_GENDER VARCHAR(5),   AMT_CREDIT FLOAT );
Start time	Tue Jul 18 10:18:22 CLT 2023
Finish time	Tue Jul 18 10:18:22 CLT 2023

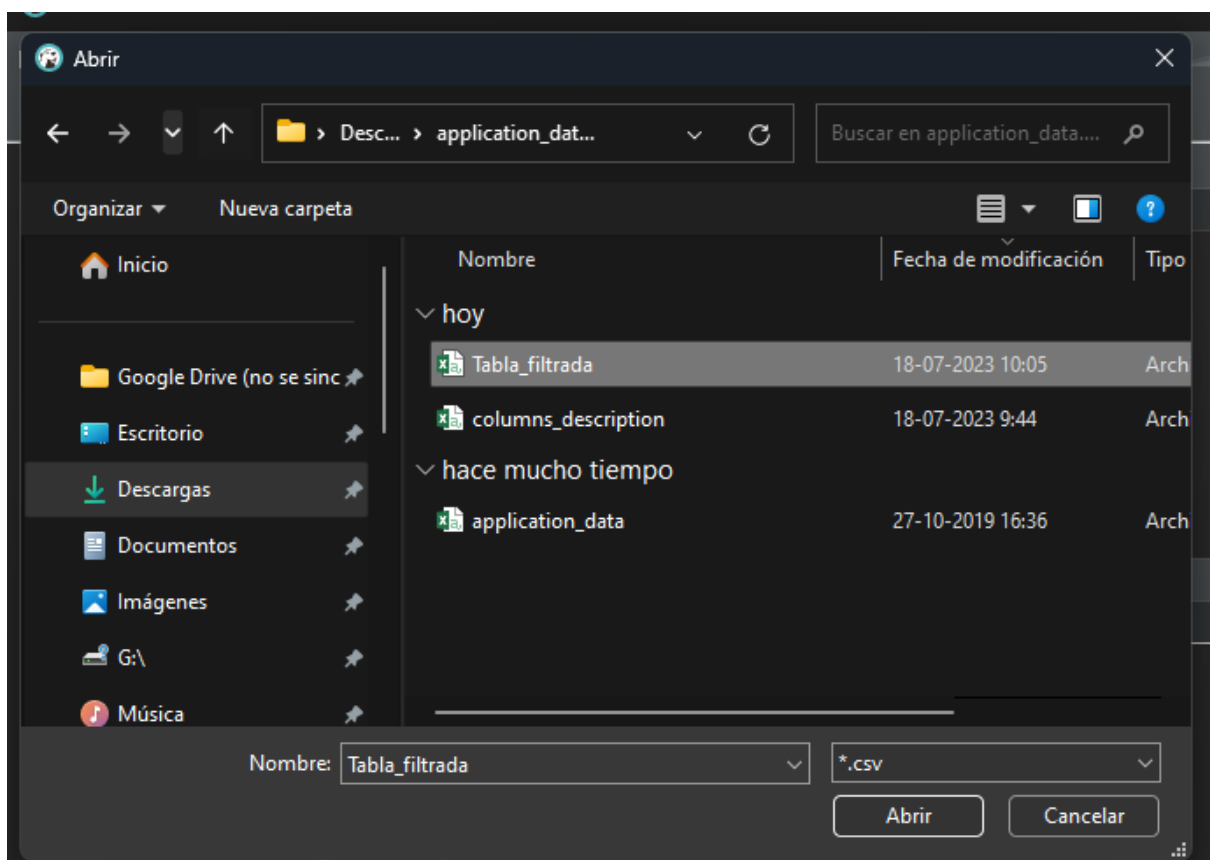
**22.** Ahora para insertar nuestros registros podemos elegir nuestra tabla > click derecho > Import data:



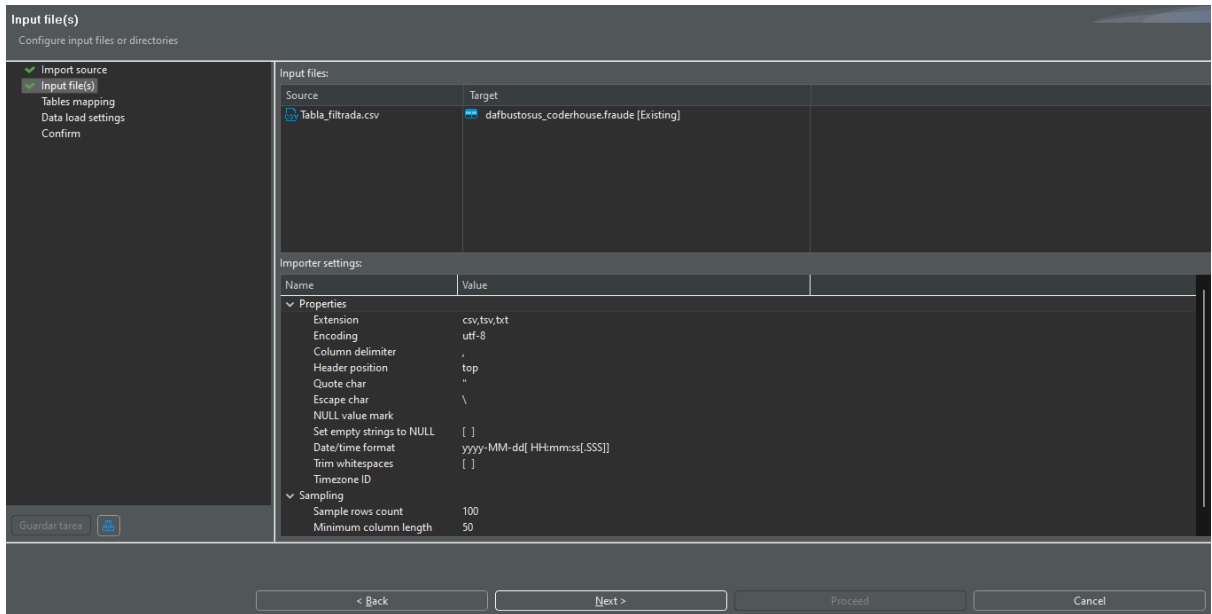
23. Aparecerá esta ventana, damos click en CSV y Next:



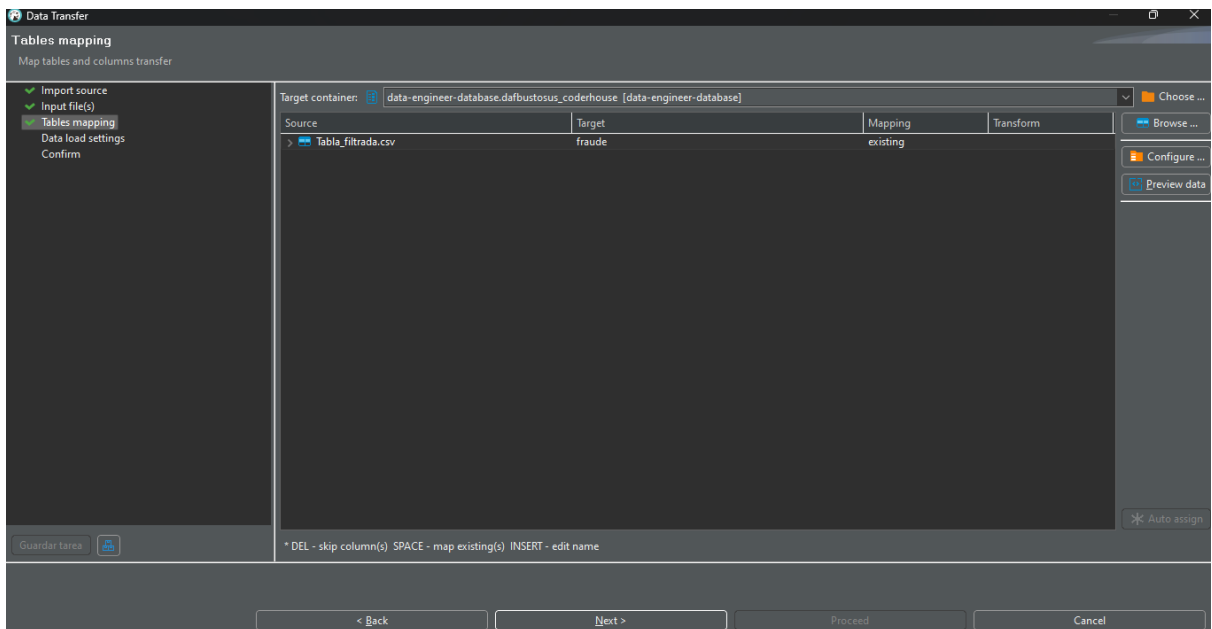
24. Luego elegimos nuestra fuente de datos:



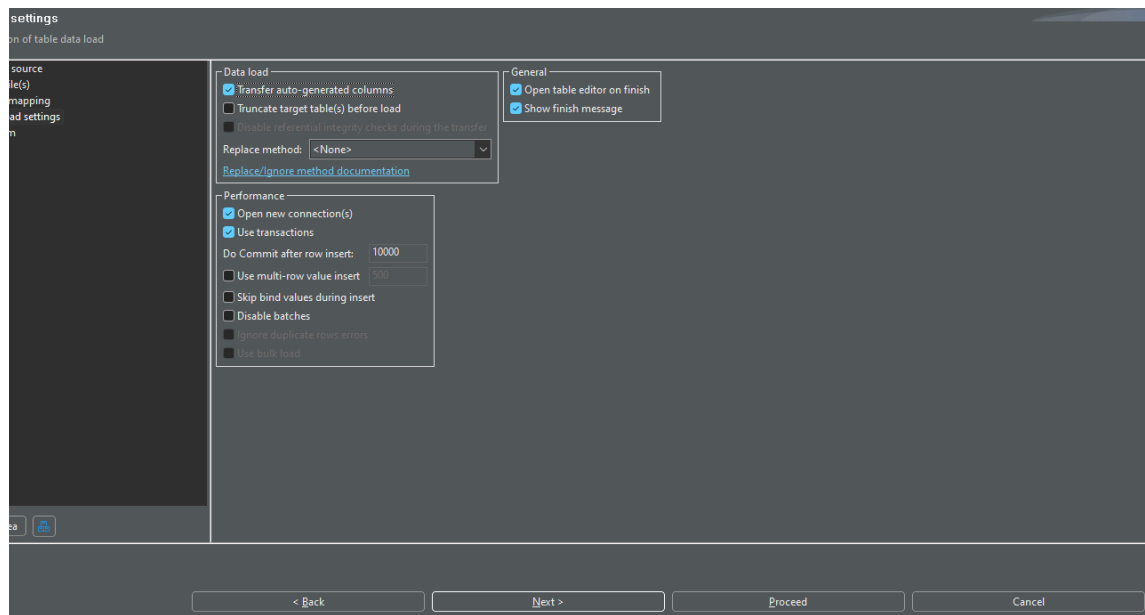




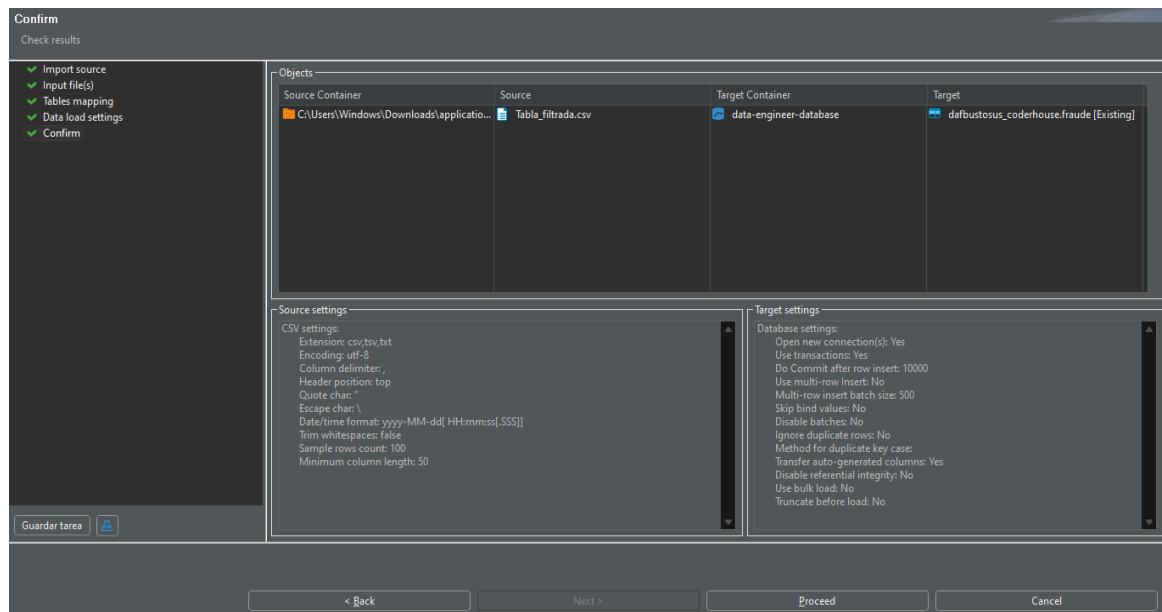
25. Elegimos la tabla de destino que en este caso se llama fraude y damos en Next:



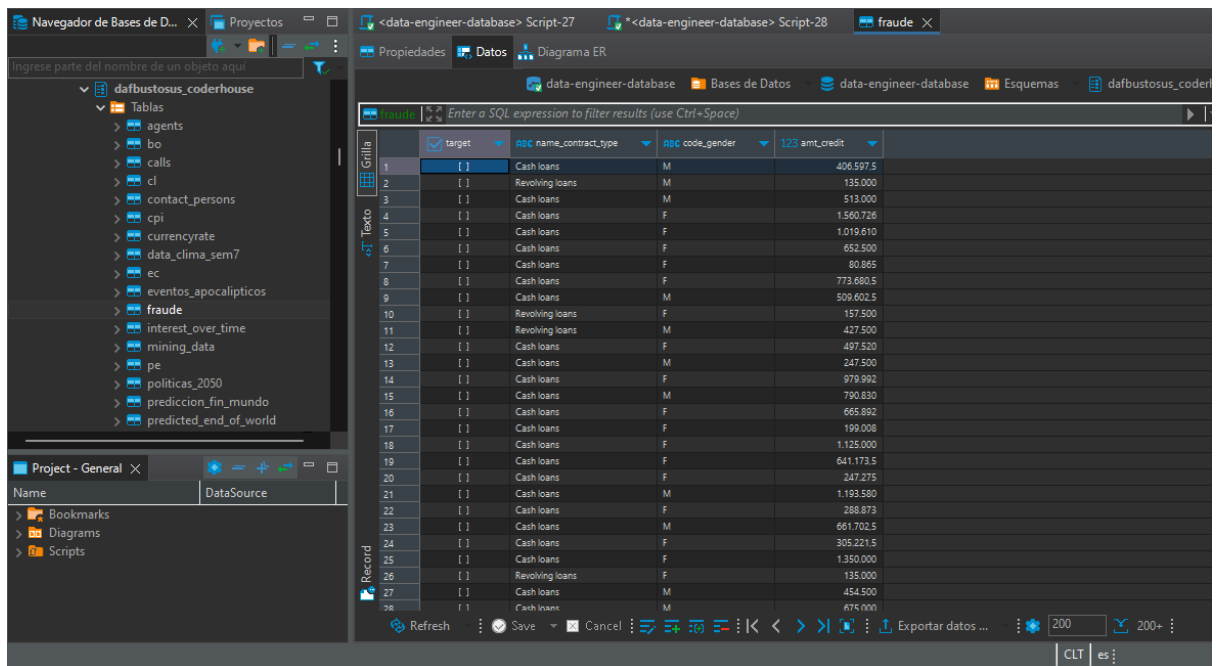
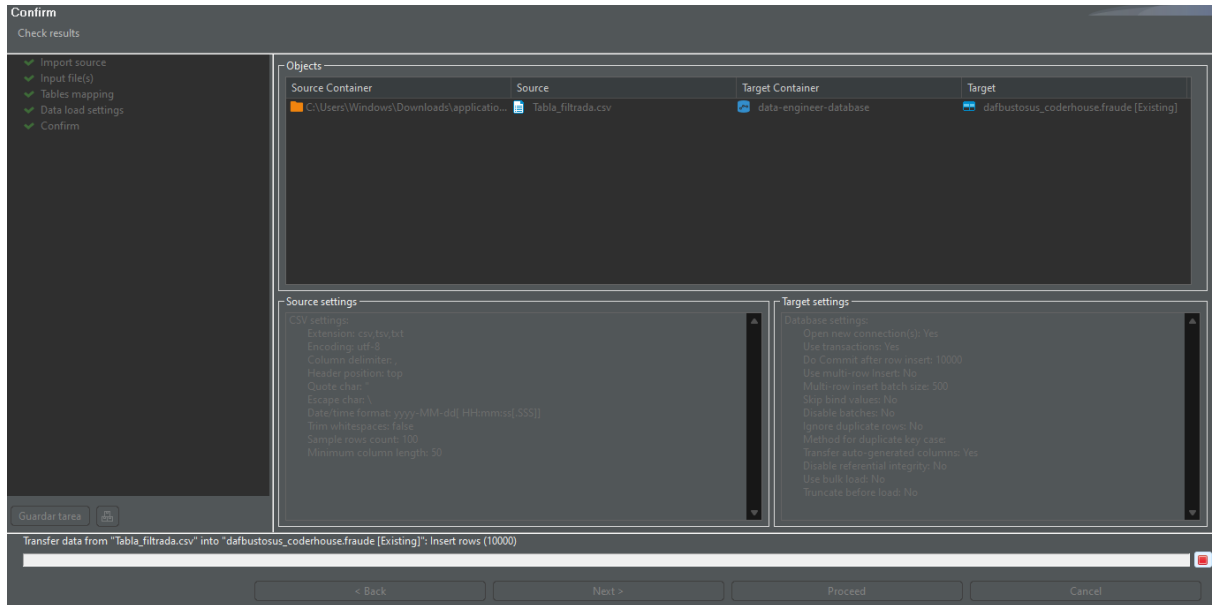
26. Dejamos la configuración por default y damos en Next:



27. Finalmente damos click en Proceed y esperamos a que carguen los datos:



28. Esto probablemente tome bastante tiempo así que ten paciencia, ya que como recordarás que Amazon Redshift es una base de datos columnar por ende es bastante lenta para cargar información de transacciones.



29. Ahora hacemos las mismas consultas que habíamos hecho en la base de datos PostgreSQL y comparemos los tiempos de ejecución

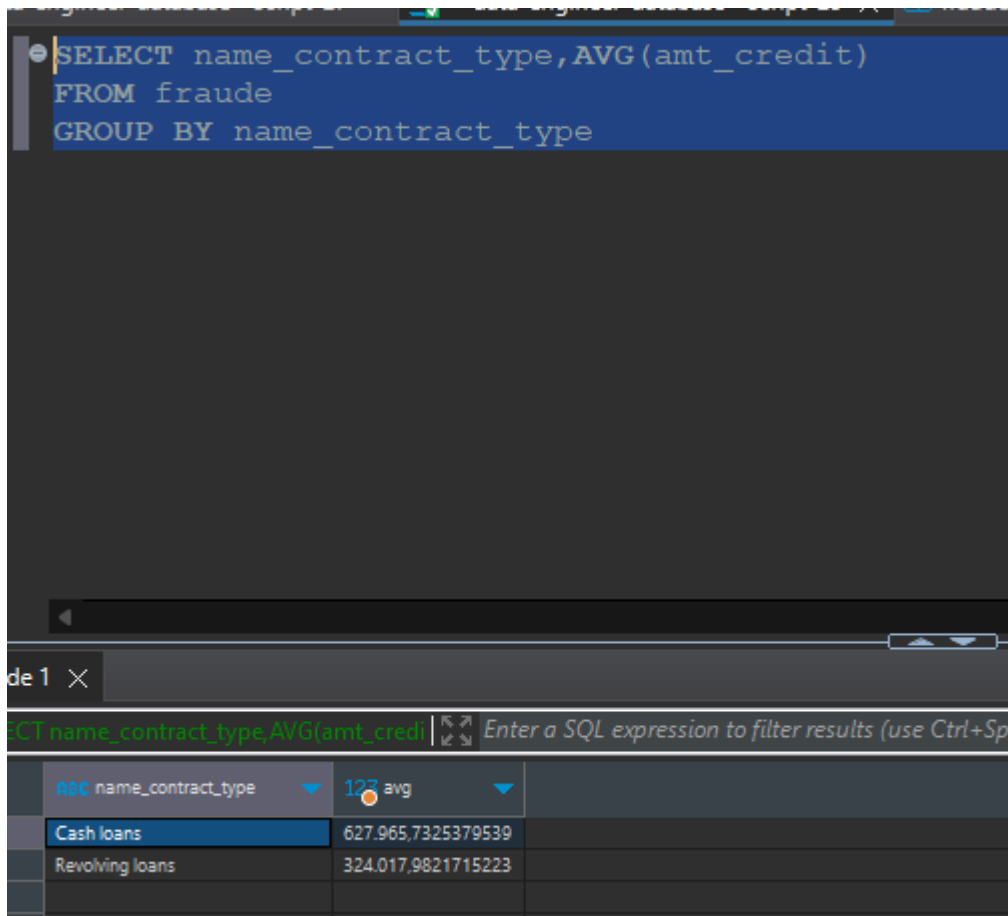
```
sql
```

```
SELECT name_contract_type,AVG(amt_credit)
```

```
FROM fraude
```

```
GROUP BY name_contract_type
```

```
---
```



The screenshot shows a SQL IDE with a query editor at the top and a results pane at the bottom. The query is:

```
SELECT name_contract_type,AVG(amt_credit)
FROM fraude
GROUP BY name_contract_type
```

The results pane shows a table with two columns: 'name\_contract\_type' and 'avg'. The data is as follows:

name_contract_type	avg
Cash loans	627.965,7325379539
Revolving loans	324.017,9821715223

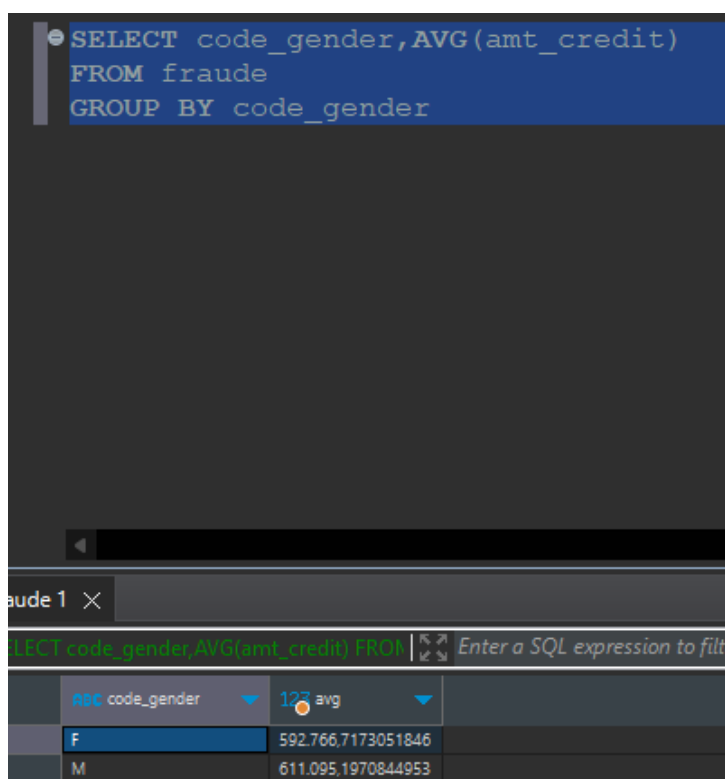
Aquí se demora 1.6 seg

```sql

```
SELECT code_gender,AVG(amt_credit)
```

```
FROM fraude
```

```
GROUP BY code_gender
```



The screenshot shows a SQL IDE with a query editor at the top and a results pane at the bottom. The query is:

```
SELECT code_gender,AVG(amt_credit)
FROM fraude
GROUP BY code_gender
```

The results pane shows a table with two columns: 'code\_gender' and 'avg'. The data is as follows:

| code_gender | avg                |
|-------------|--------------------|
| F           | 592.766,7173051846 |
| M           | 611.095,1970844953 |

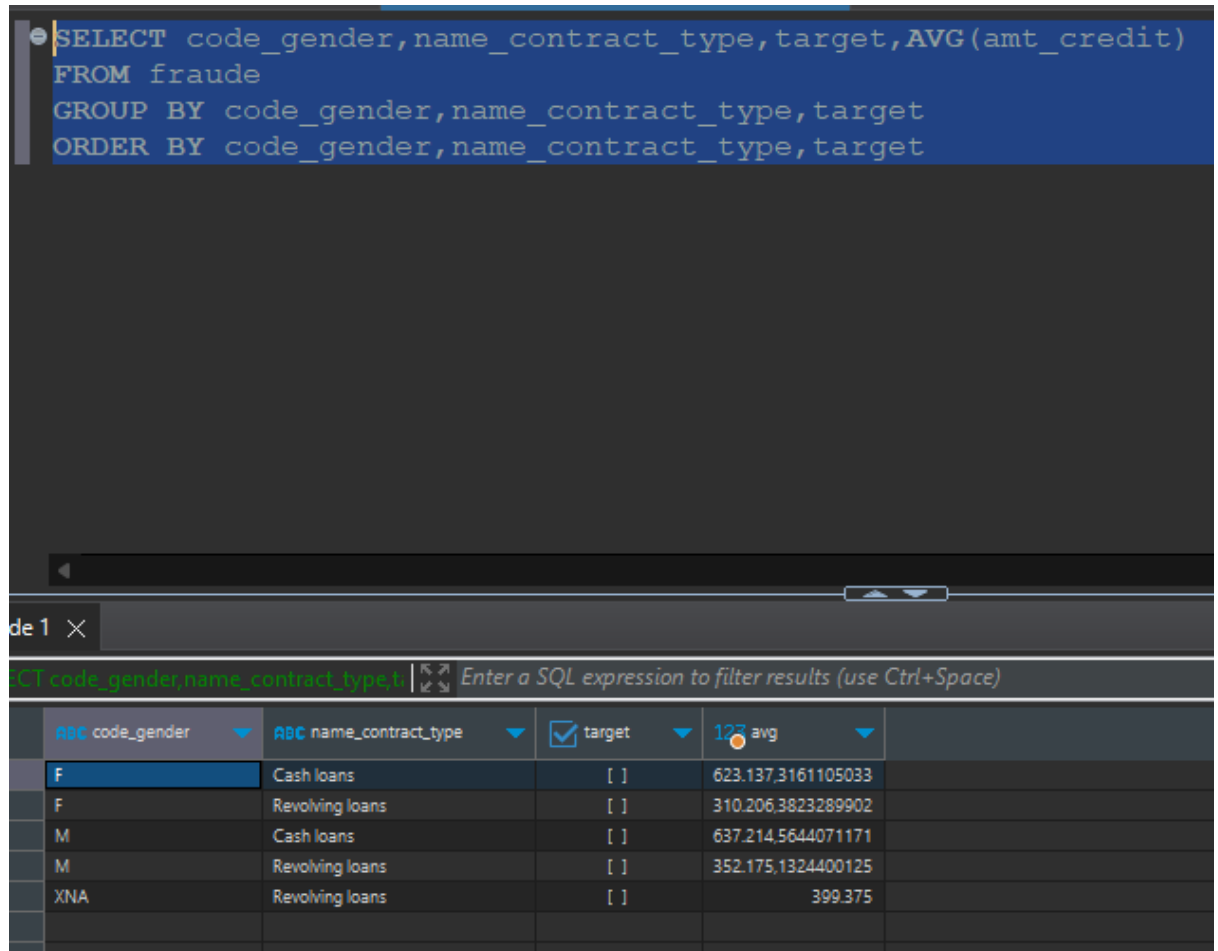
```

2.3 seg en esta

```sql

```
SELECT code_gender,name_contract_type,target,AVG(amt_credit)
FROM fraude
GROUP BY code_gender,name_contract_type,target
ORDER BY code_gender,name_contract_type,target
```

```



En esta última se demora solo 6.1 seg

**30.** En resumen tenemos esto:

	Amazon RD (seg)	PostgreSQL (seg)
Consulta 1	1.6	6.9
Consulta 2	2.3	3.8
Consulta 3	6.1	17.7

**Conclusión a presentarte a tu Data Leader:** Claramente se ve que Amazon RD es mucho más rápido para consultas analíticas que involucran agrupaciones en comparación con bases de datos como PostgreSQL que son mucho mejores para sistemas transaccionales de inserción de filas.