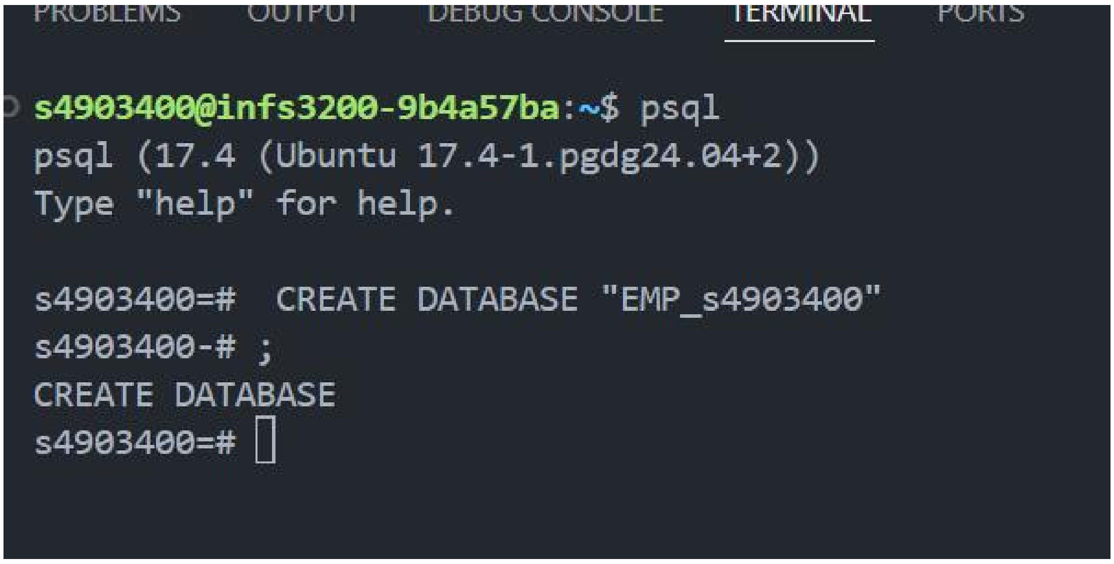
Task 1: Load Database

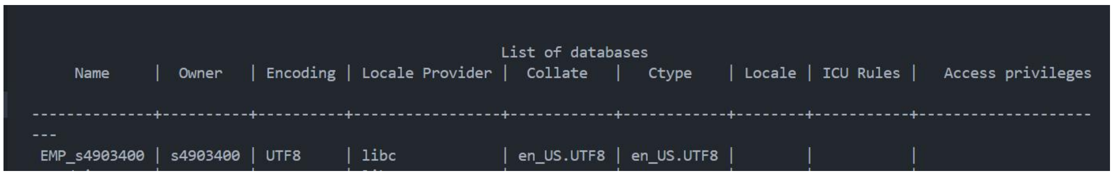
Create a database named 'EMP_s1234567' (1234567 should be replaced by your studentID).


PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

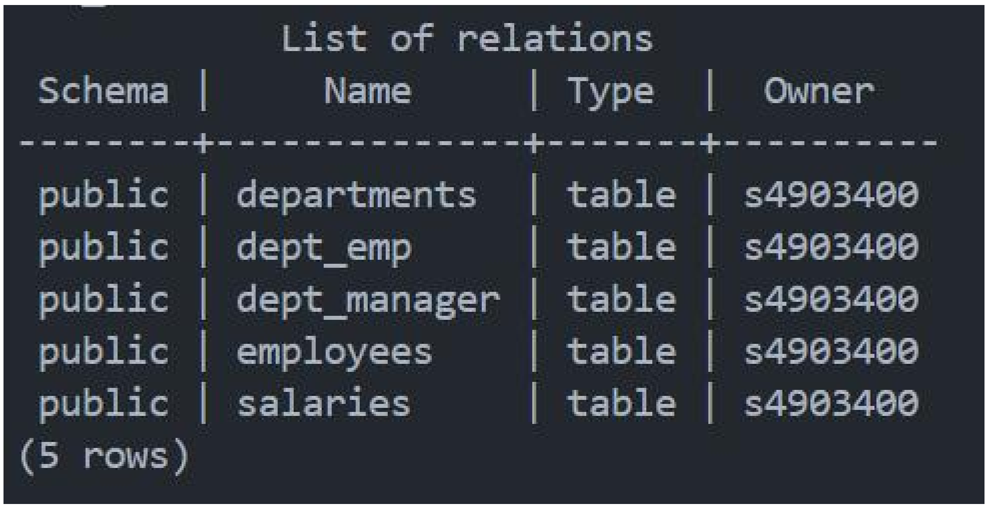
```
s4903400@infs3200-9b4a57ba:~$ psql
psql (17.4 (Ubuntu 17.4-1.pgdg24.04+2))
Type "help" for help.

s4903400=#  CREATE DATABASE "EMP_s4903400"
s4903400-# ;
CREATE DATABASE
s4903400=# 
```

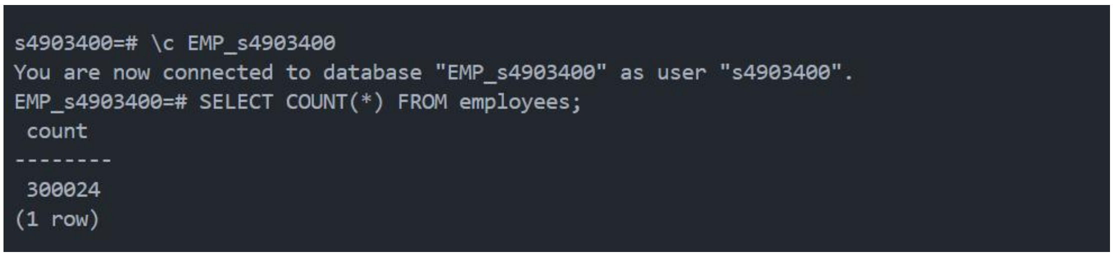
Load the database with the provided sql files.


```
                                                  List of databases
     Name      |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | Locale | ICU Rules |   Access privileges
-------------+---------+---------+----------------+-----------+-----------+--------+----------+-------------------
---
 EMP_s4903400 | s4903400 | UTF8     | libc            | en_US.UTF8 | en_US.UTF8 |        |           |
```

You should provide some screenshots to list the databaseand tables, after creating all necessary tables and databases.

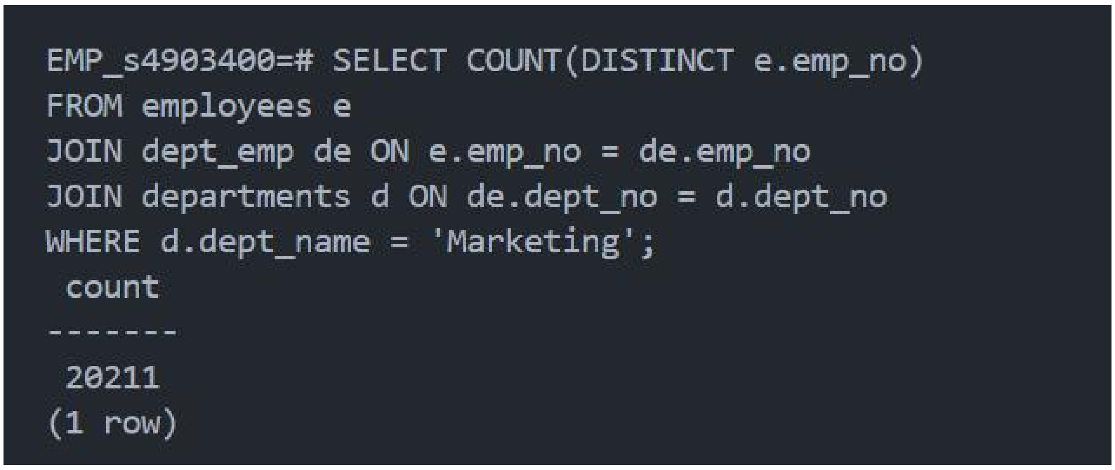

```
              List of relations
 Schema |      Name      | Type  |  Owner
--------+---------------+-------+----------
 public | departments   | table | s4903400
 public | dept_emp      | table | s4903400
 public | dept_manager  | table | s4903400
 public | employees     | table | s4903400
 public | salaries      | table | s4903400
(5 rows)
```

Write sql queries to count the number of entries in table 'employees'.



```
s4903400=# \c EMP_s4903400
You are now connected to database "EMP_s4903400" as user "s4903400".
EMP_s4903400=# SELECT COUNT(*) FROM employees;
 count
--------
 300024
(1 row)
```

Write sql queries to count the number of employees from the 'Marketing' department

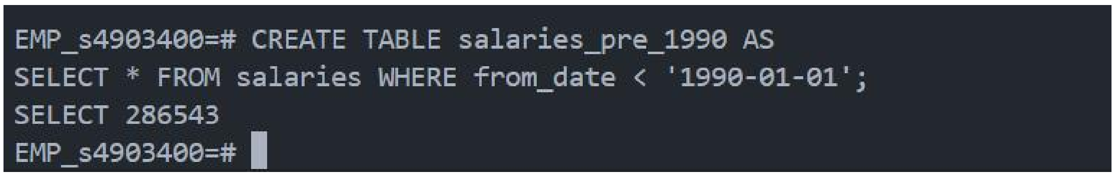


```
EMP_s4903400=# SELECT COUNT(DISTINCT e.emp_no)
FROM employees e
JOIN dept_emp de ON e.emp_no = de.emp_no
JOIN departments d ON de.dept_no = d.dept_no
WHERE d.dept_name = 'Marketing';
 count
-------
 20211
(1 row)
```

Task 2: Database Fragmentation

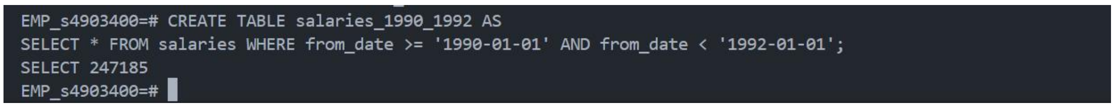
Write sql queries to perform horizontal fragmentation on table 'salaries', based on the following rules:

'from_date' before 1990-01-01



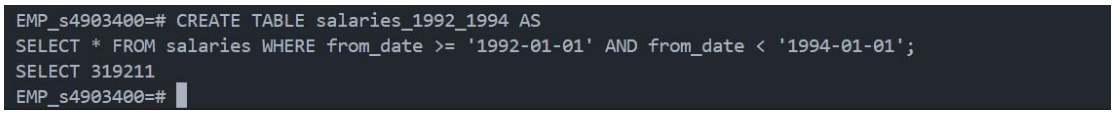
```
EMP_s4903400=# CREATE TABLE salaries_pre_1990 AS
SELECT * FROM salaries WHERE from_date < '1990-01-01';
SELECT 286543
EMP_s4903400=#
```

'from_date' no earlier than 1990-01-01 and before 1992-01-01



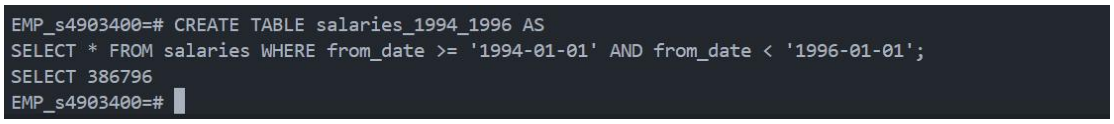
```
EMP_s4903400=# CREATE TABLE salaries_1990_1992 AS
SELECT * FROM salaries WHERE from_date >= '1990-01-01' AND from_date < '1992-01-01';
SELECT 247185
EMP_s4903400=#
```

'from_date' no earlier than 1992-01-01 and before 1994-01-01



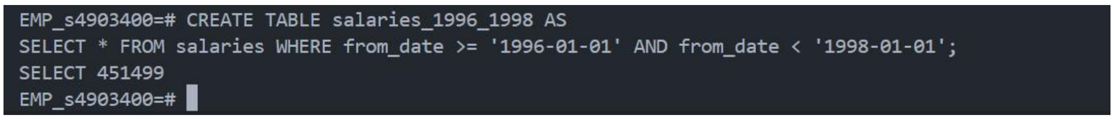
```
EMP_s4903400=# CREATE TABLE salaries_1992_1994 AS
SELECT * FROM salaries WHERE from_date >= '1992-01-01' AND from_date < '1994-01-01';
SELECT 319211
EMP_s4903400=#
```

'from_date' no earlier than 1994-01-01 and before 1996-01-01



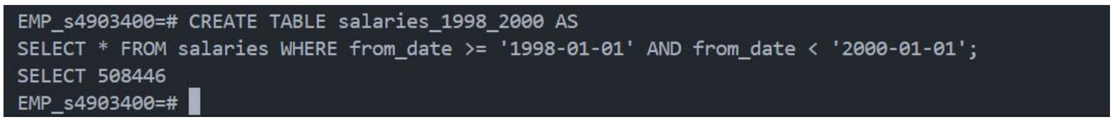
```
EMP_s4903400=# CREATE TABLE salaries_1994_1996 AS
SELECT * FROM salaries WHERE from_date >= '1994-01-01' AND from_date < '1996-01-01';
SELECT 386796
EMP_s4903400=#
```

'from_date' no earlier than 1996-01-01 and before 1998-01-01



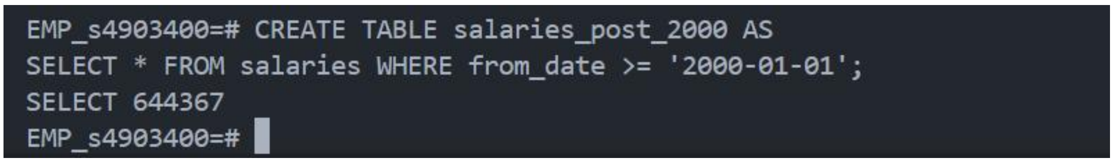
```
EMP_s4903400=# CREATE TABLE salaries_1996_1998 AS
SELECT * FROM salaries WHERE from_date >= '1996-01-01' AND from_date < '1998-01-01';
SELECT 451499
EMP_s4903400=#
```

'from_date' no earlier than 1998-01-01 and before 2000-01-01



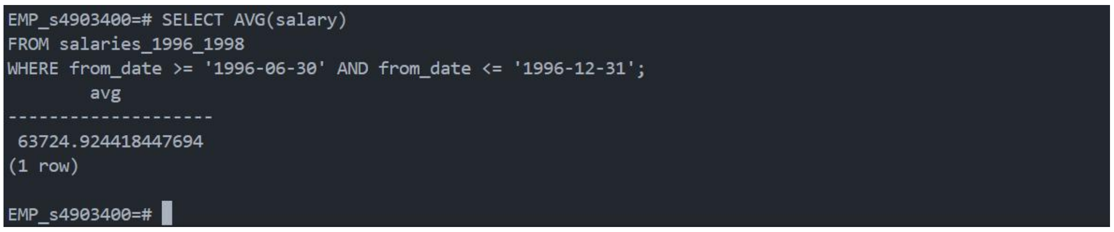
```
EMP_s4903400=# CREATE TABLE salaries_1998_2000 AS
SELECT * FROM salaries WHERE from_date >= '1998-01-01' AND from_date < '2000-01-01';
SELECT 508446
EMP_s4903400=#
```

'from_date' no earlier than 2000-01-01



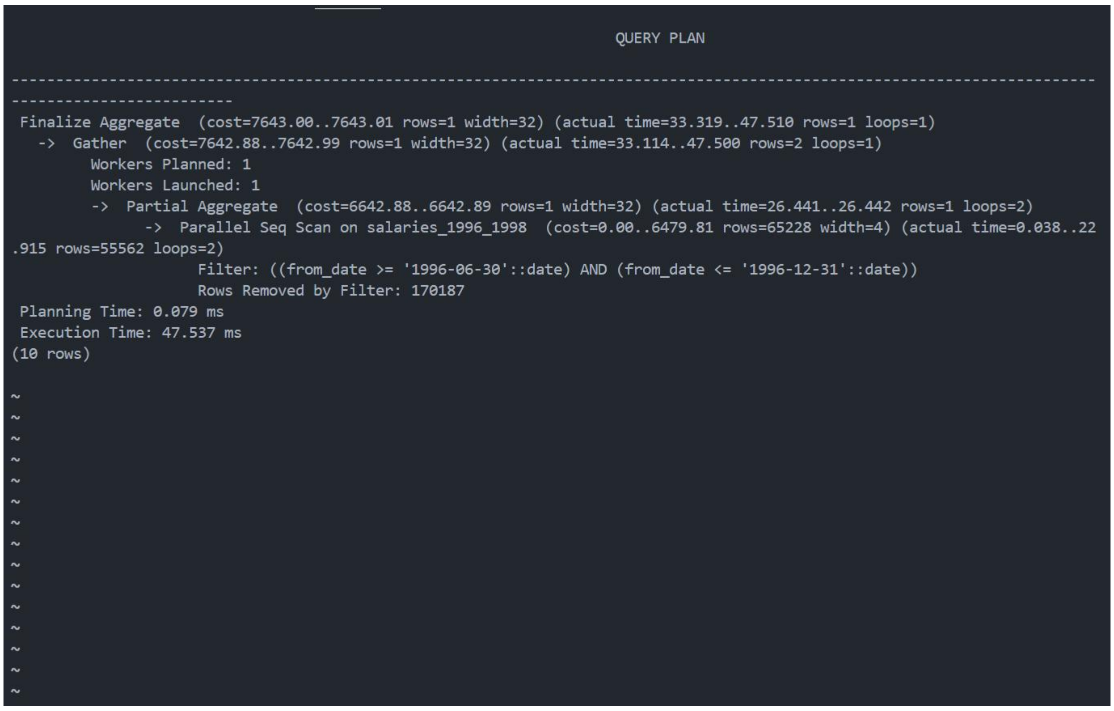
```
EMP_s4903400=# CREATE TABLE salaries_post_2000 AS
SELECT * FROM salaries WHERE from_date >= '2000-01-01';
SELECT 644367
EMP_s4903400=#
```

Calculate the average employee salary in between 1996-06-30 and 1996-12-31 of the attribute “from_date” on the fragmented ‘salaries’ table. Show the query explanation.



```
EMP_s4903400=# SELECT AVG(salary)
FROM salaries_1996_1998
WHERE from_date >= '1996-06-30' AND from_date <= '1996-12-31';
        avg
--------------------
 63724.924418447694
(1 row)

EMP_s4903400=#
```



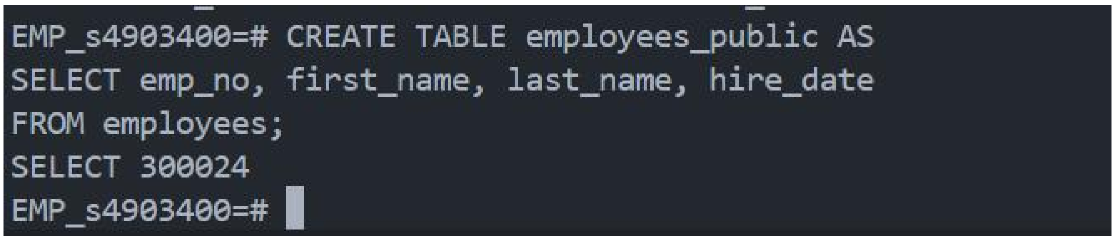
```
                                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
------------------------
 Finalize Aggregate  (cost=7643.00..7643.01 rows=1 width=32) (actual time=33.319..47.510 rows=1 loops=1)
   ->  Gather  (cost=7642.88..7642.99 rows=1 width=32) (actual time=33.114..47.500 rows=2 loops=1)
         Workers Planned: 1
         Workers Launched: 1
         ->  Partial Aggregate  (cost=6642.88..6642.89 rows=1 width=32) (actual time=26.441..26.442 rows=1 loops=2)
               ->  Parallel Seq Scan on salaries_1996_1998  (cost=0.00..6479.81 rows=65228 width=4) (actual time=0.038..22
.915 rows=55562 loops=2)
                     Filter: ((from_date >= '1996-06-30'::date) AND (from_date <= '1996-12-31'::date))
                     Rows Removed by Filter: 170187
 Planning Time: 0.079 ms
 Execution Time: 47.537 ms
(10 rows)

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Explanation：

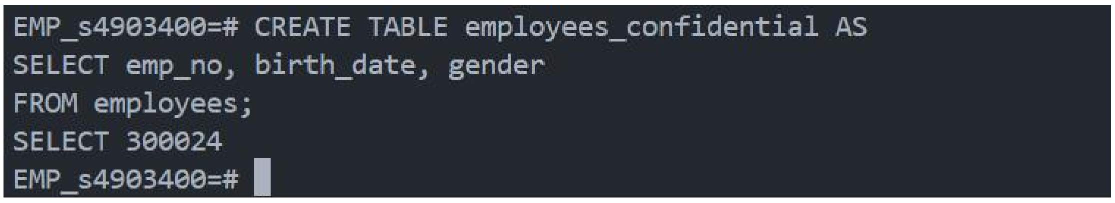
For the query SELECT AVG(salary) FROM salaries_1996_1998 WHERE from_date >= '1996-06-30' AND from_date <= '1996-12-31', PostgreSQL processes the salaries_1996_1998 table using a parallel sequential scan (Parallel Seq Scan). The execution plan shows that PSQL launches one additional worker (a total of two), with each worker handling approximately 225,749 rows. During the scan, the condition (from_date >= '1996-06-30' AND from_date <= '1996-12-31') filters out 170,187 rows per worker, retaining 55,562 rows, and this step takes about 22.915 ms. Subsequently, each worker performs a partial aggregation, summing the salary values and counting the rows for its 55,562 rows, which takes around 26.44 ms. The main process gathers the results from the two workers in about 47.500 ms and completes the final aggregation in 47.510 ms to calculate the average, with a total execution time of 47.537 ms.

Write sql queries to perform vertical fragmentation on table 'employee', based on the following rules:
'emp_no', 'first_name', 'last_name', and 'hire_date' should be stored in table 'employees_public'



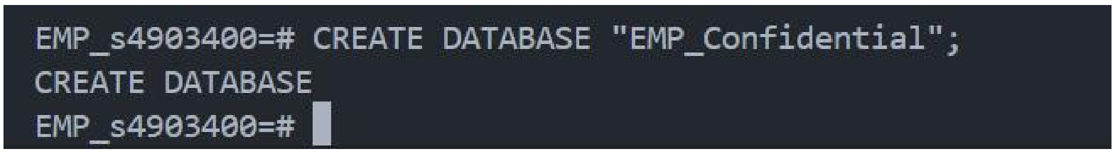
```
EMP_s4903400=# CREATE TABLE employees_public AS
SELECT emp_no, first_name, last_name, hire_date
FROM employees;
SELECT 300024
EMP_s4903400=#
```

'emp_no', 'birth_date', and 'gender' should be stored in table 'employees_ confidential'

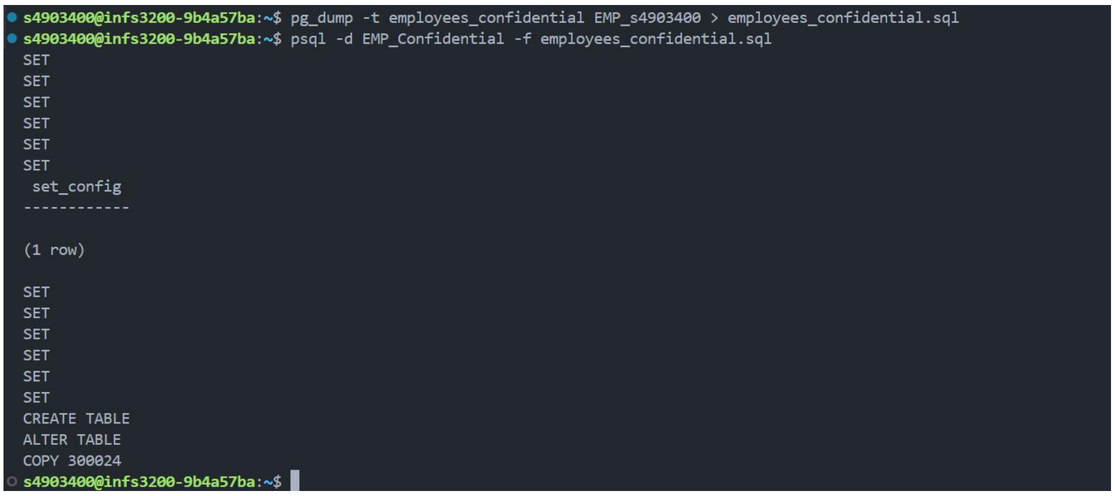


```
EMP_s4903400=# CREATE TABLE employees_confidential AS
SELECT emp_no, birth_date, gender
FROM employees;
SELECT 300024
EMP_s4903400=#
```

The 'employees_ confidential' table should be stored in a new database called 'EMP_Confidential'.



```
EMP_s4903400=# CREATE DATABASE "EMP_Confidential";
CREATE DATABASE
EMP_s4903400=#
```



```
s4903400@infs3200-9b4a57ba:~$ pg_dump -t employees_confidential EMP_s4903400 > employees_confidential.sql
s4903400@infs3200-9b4a57ba:~$ psql -d EMP_Confidential -f employees_confidential.sql
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
COPY 300024
s4903400@infs3200-9b4a57ba:~$
```

Task 3: Database Replication

Q1:

1.Full copy

Advantages:

High data integrity: Each server has complete data.

High availability: When one server fails, the other can take over.

Cons:

High storage cost: Each server stores complete data, consuming storage resources.

Complex synchronization: Data is synchronized between multiple servers, making it difficult to maintain consistency.

2.Partial replication
Advantages:
Low storage cost: Each server only stores part of the data.
Synchronization is relatively simple: it only needs to be synchronized between related servers.
Cons:
Limited data integrity: data on a single server is incomplete, and multi-server collaboration is required for cross-shard queries.
Possible performance bottleneck: There is a network delay for cross-server queries.

3.Not reproduce
Advantages:
Lowest storage cost: Only the primary server stores data.
Simple and easy maintenance: No data synchronization issues.
Cons:
Single point of failure risk: The primary server fails and the system may fail.
Limited performance: All operations are concentrated on the primary server, which is prone to performance bottlenecks.

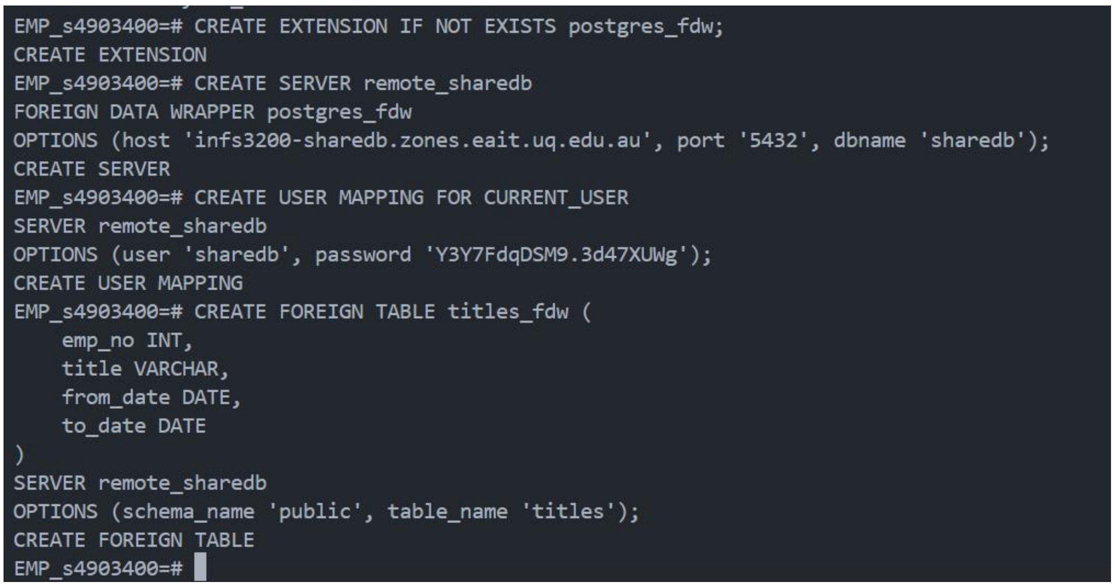
Q2:
Determine the shard location: According to the 'emp_no' value, use the shard key algorithm to determine the shard number.

Locate the server: Locate the server that stores the fragment based on the fragment allocation scheme.

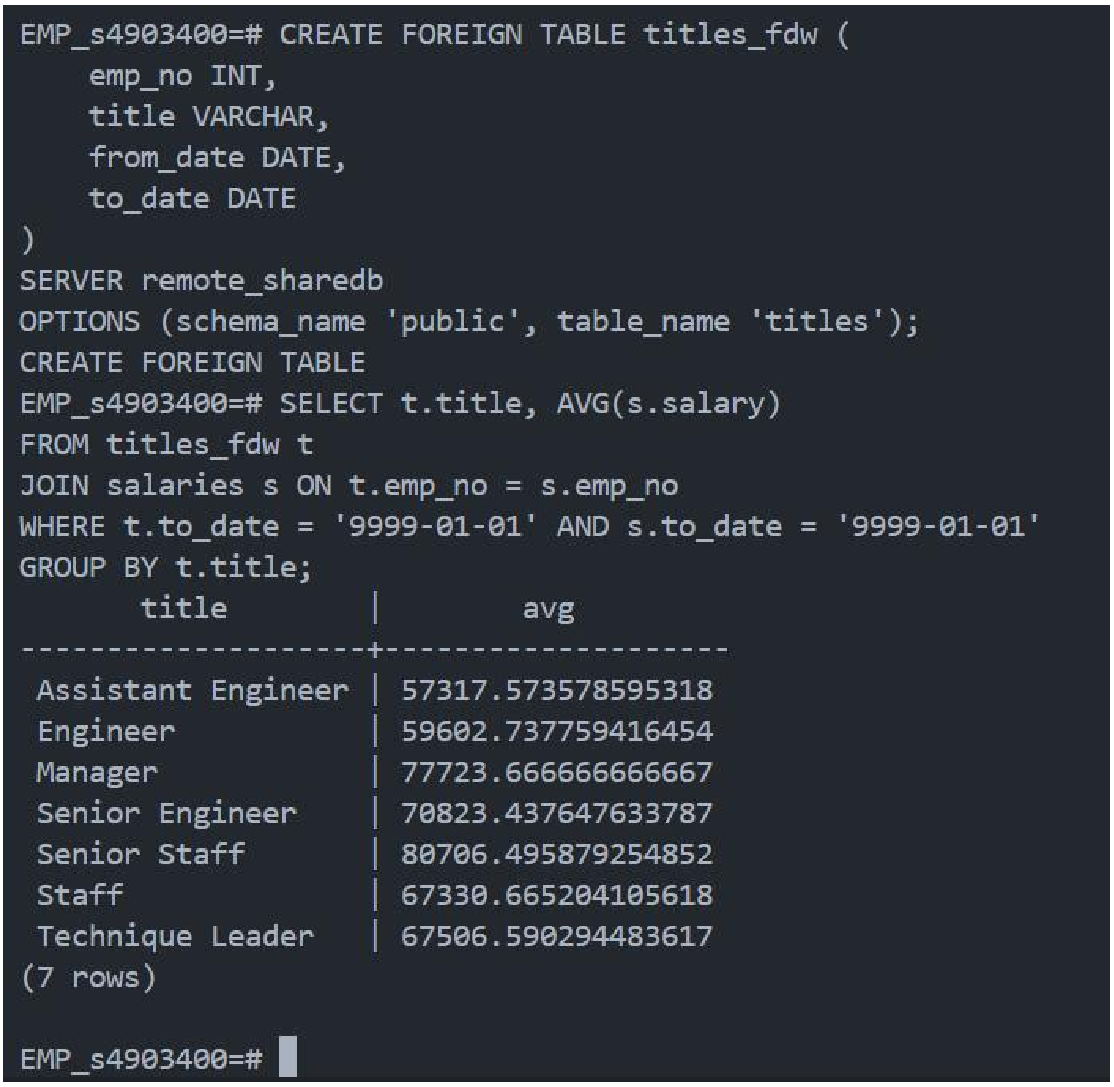
Send update request: Send the update request to the corresponding server to perform the update.

Task 4: Access foreign data with FDW

The table 'titles' is stored in a remote database that requires you to use Foreign Data Wrappers (FDW) to access the data. You are provided with the following login details:
➢ User Name: sharedb
➢ Password: Y3Y7FdqDSM9.3d47XUWg
➢ Host: infs3200-sharedb.zones.eait.uq.edu.au
➢ Port: 5432
➢ Database Name: sharedb
(1) Write sql queries to establish a FDW to the external DB. Queries/commands should be provided.


```
EMP_s4903400=# CREATE EXTENSION IF NOT EXISTS postgres_fdw;
CREATE EXTENSION
EMP_s4903400=# CREATE SERVER remote_sharedb
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'infs3200-sharedb.zones.eait.uq.edu.au', port '5432', dbname 'sharedb');
CREATE SERVER
EMP_s4903400=# CREATE USER MAPPING FOR CURRENT_USER
SERVER remote_sharedb
OPTIONS (user 'sharedb', password 'Y3Y7FdqDSM9.3d47XUWg');
CREATE USER MAPPING
EMP_s4903400=# CREATE FOREIGN TABLE titles_fdw (
    emp_no INT,
    title VARCHAR,
    from_date DATE,
    to_date DATE
)
SERVER remote_sharedb
OPTIONS (schema_name 'public', table_name 'titles');
CREATE FOREIGN TABLE
EMP_s4903400=# 
```

(1) For each unique 'title' in table 'titles', calculate the averaged current salary. You should query from the foreign table and provide the queries and screenshots of the outputs.
Hint: current staff has 'to_date' that equals to '9999-01-01'



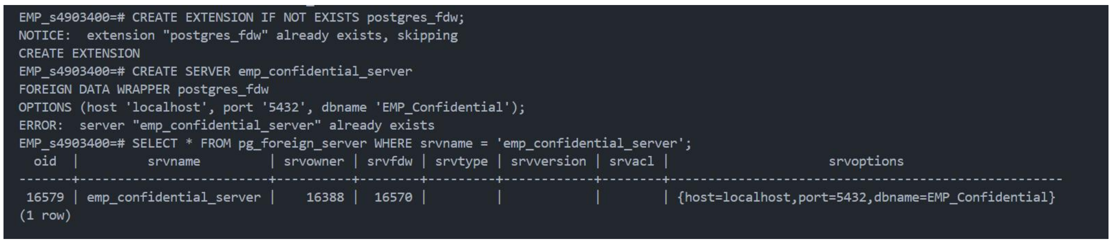
```
EMP_s4903400=# CREATE FOREIGN TABLE titles_fdw (
    emp_no INT,
    title VARCHAR,
    from_date DATE,
    to_date DATE
)
SERVER remote_sharedb
OPTIONS (schema_name 'public', table_name 'titles');
CREATE FOREIGN TABLE
EMP_s4903400=# SELECT t.title, AVG(s.salary)
FROM titles_fdw t
JOIN salaries s ON t.emp_no = s.emp_no
WHERE t.to_date = '9999-01-01' AND s.to_date = '9999-01-01'
GROUP BY t.title;
       title        |        avg
--------------------+--------------------
 Assistant Engineer | 57317.573578595318
 Engineer           | 59602.737759416454
 Manager            | 77723.666666666667
 Senior Engineer    | 70823.437647633787
 Senior Staff       | 80706.495879254852
 Staff              | 67330.665204105618
 Technique Leader   | 67506.590294483617
(7 rows)

EMP_s4903400=#
```

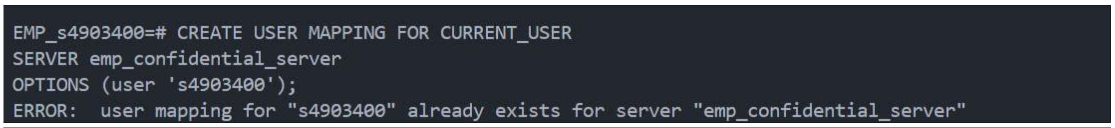
(1) Consider that the current employee table is vertically fragmented and stored on different databases. Perform semi join from 'EMP' database to select the last name and first name of employees that 'birth_date' is no earlier than 1970-01-01 and before 1975-01- 01. You should create another foreign table mapping to the necessary table and access the vertical fragmentation table through FDW. Queries, screenshots and brief descriptions are required.

Note: 'birth_date' information can only be accessed from 'EMP_Confidential' database through FDW. Direct read from the original employee table will receive 0 mark for this question.

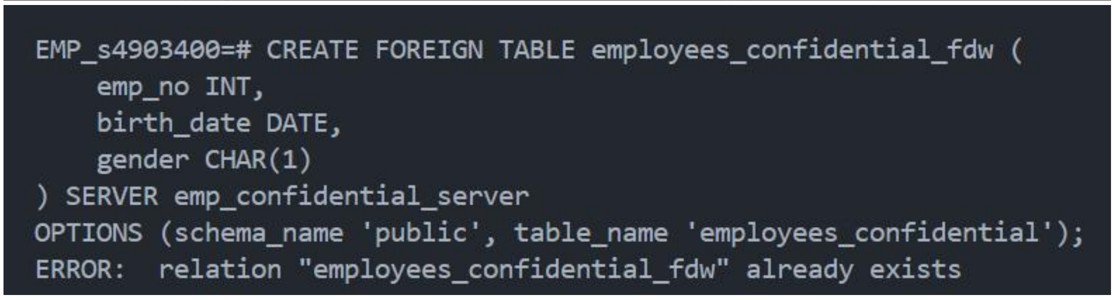
Note： I successfully created the FDW extension and server emp_confidential_server, fixed the user mapping, and set the remote user s4903400 and password s4903400Pass to ensure a connection to EMP_Confidential.



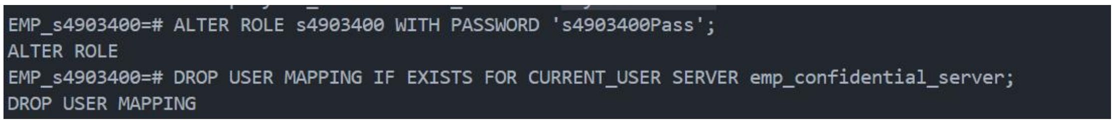
```
EMP_s4903400=# CREATE EXTENSION IF NOT EXISTS postgres_fdw;
NOTICE:  extension "postgres_fdw" already exists, skipping
CREATE EXTENSION
EMP_s4903400=# CREATE SERVER emp_confidential_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', port '5432', dbname 'EMP_Confidential');
ERROR:  server "emp_confidential_server" already exists
EMP_s4903400=# SELECT * FROM pg_foreign_server WHERE srvname = 'emp_confidential_server';
  oid  |         srvname         | srvowner | srvfdw | srvtype | srvversion | srvacl |                      srvoptions
-------+-------------------------+----------+--------+---------+------------+--------+------------------------------------------------------
 16579 | emp_confidential_server |    16388 |  16570 |         |            |        | {host=localhost,port=5432,dbname=EMP_Confidential}
(1 row)
```



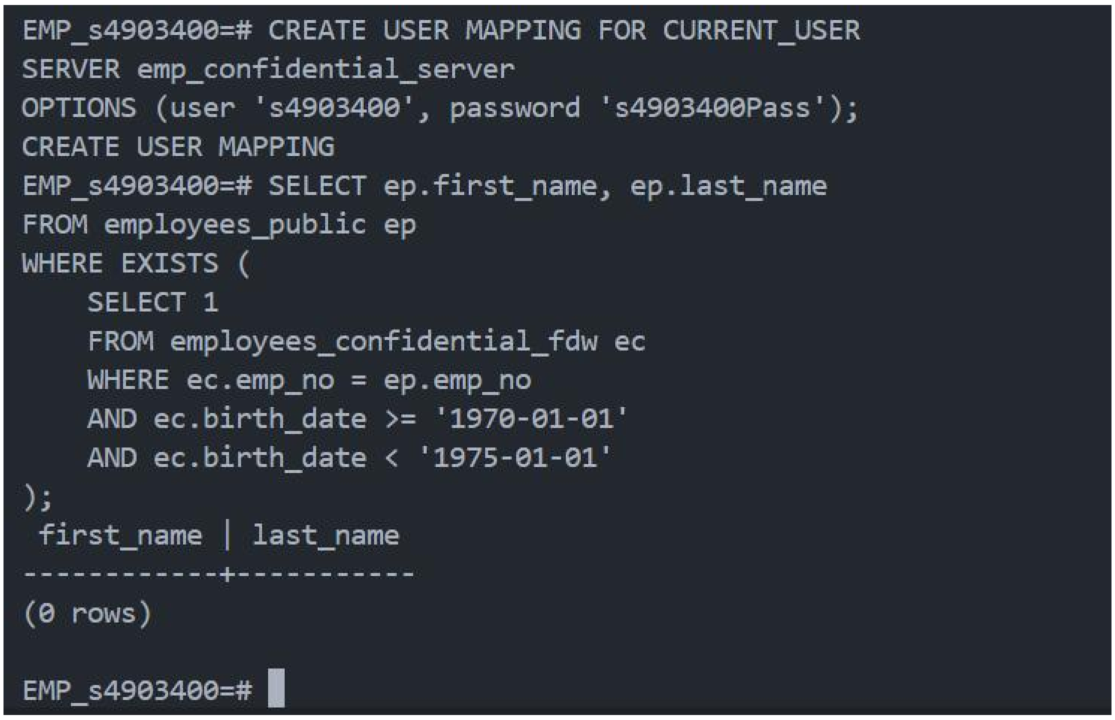
```
EMP_s4903400=# CREATE USER MAPPING FOR CURRENT_USER
SERVER emp_confidential_server
OPTIONS (user 's4903400');
ERROR:  user mapping for "s4903400" already exists for server "emp_confidential_server"
```



```
EMP_s4903400=# CREATE FOREIGN TABLE employees_confidential_fdw (
    emp_no INT,
    birth_date DATE,
    gender CHAR(1)
) SERVER emp_confidential_server
OPTIONS (schema_name 'public', table_name 'employees_confidential');
ERROR:  relation "employees_confidential_fdw" already exists
```



```
EMP_s4903400=# ALTER ROLE s4903400 WITH PASSWORD 's4903400Pass';
ALTER ROLE
EMP_s4903400=# DROP USER MAPPING IF EXISTS FOR CURRENT_USER SERVER emp_confidential_server;
DROP USER MAPPING
```



```
EMP_s4903400=# CREATE USER MAPPING FOR CURRENT_USER
SERVER emp_confidential_server
OPTIONS (user 's4903400', password 's4903400Pass');
CREATE USER MAPPING
EMP_s4903400=# SELECT ep.first_name, ep.last_name
FROM employees_public ep
WHERE EXISTS (
    SELECT 1
    FROM employees_confidential_fdw ec
    WHERE ec.emp_no = ep.emp_no
    AND ec.birth_date >= '1970-01-01'
    AND ec.birth_date < '1975-01-01'
);
 first_name | last_name
------------+-----------
(0 rows)

EMP_s4903400=#
```

(1) Compared with inner join, will semi join incur higher transmission cost under this scenario? You should respectively show the steps for semi-join and inner-join with queries and transmission cost (not the join cost). Queries, screenshots of the query plans, and explanations about how the joins are performed and why one join strategy has more transmission cost than the other should be included in the submission.

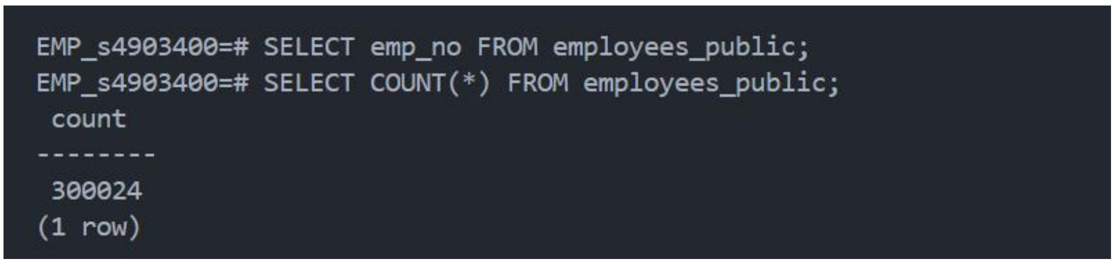
Note: Screenshots are required for the comparison.

Note:

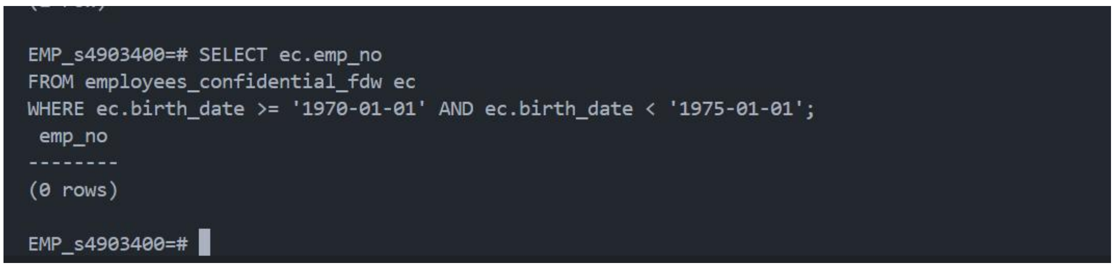
Semi join: Approximately 1,200,096 bytes (300,024 × 4 bytes), returned 0 bytes, total cost 1,200,096 bytes.

Inner Join: The objective is to transfer all the data from employees_confidential to the local system.
Fields: emp_no (4 bytes), birth_date (4 bytes), gender (1 byte).
Total transmission cost: 300,024 × 9N = 2,700,216 bytes

Difference: If a remote table column (such as birth_date) is selected, the inner join needs to transfer additional data, resulting in a higher cost; in this example, with no matches, the costs are the same.

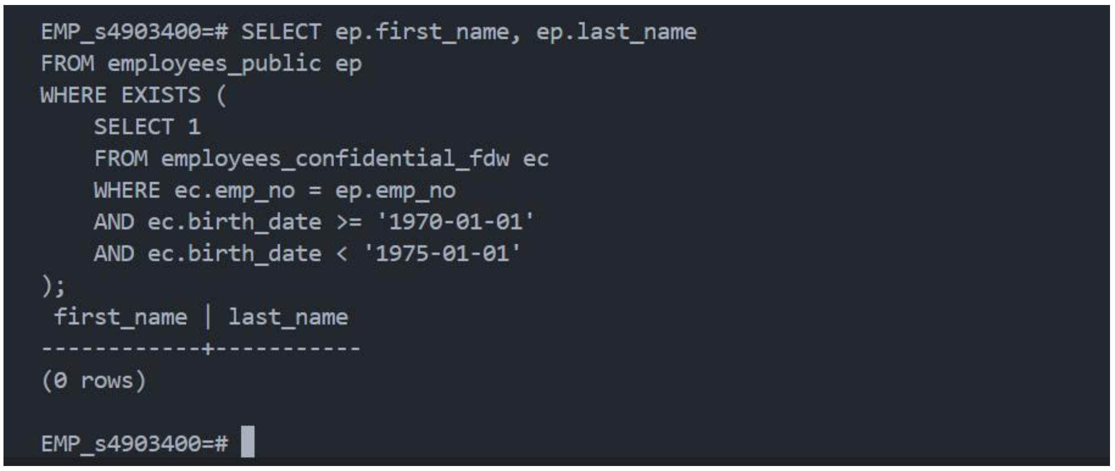


```
EMP_s4903400=# SELECT emp_no FROM employees_public;
EMP_s4903400=# SELECT COUNT(*) FROM employees_public;
 count
--------
 300024
(1 row)
```



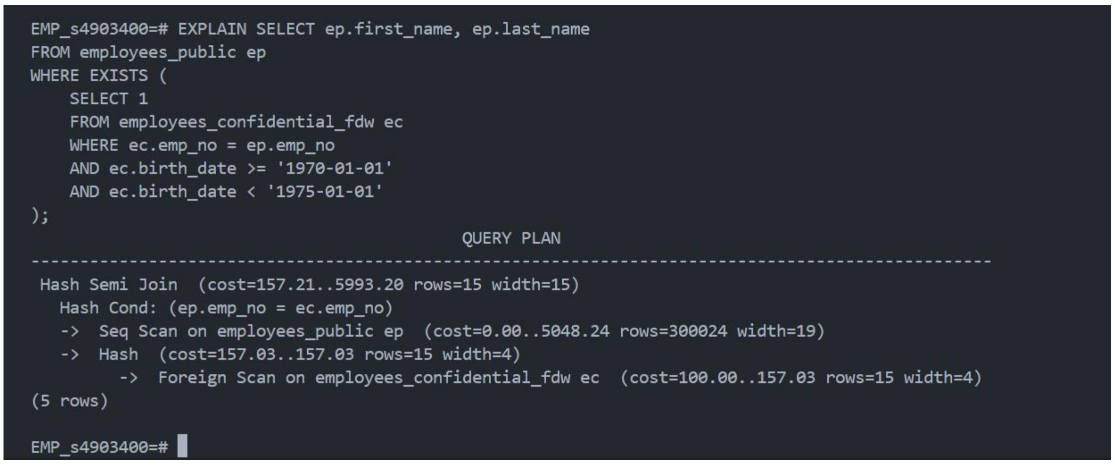
```
EMP_s4903400=# SELECT ec.emp_no
FROM employees_confidential_fdw ec
WHERE ec.birth_date >= '1970-01-01' AND ec.birth_date < '1975-01-01';
 emp_no
--------
(0 rows)

EMP_s4903400=# 
```



```
EMP_s4903400=# SELECT ep.first_name, ep.last_name
FROM employees_public ep
WHERE EXISTS (
    SELECT 1
    FROM employees_confidential_fdw ec
    WHERE ec.emp_no = ep.emp_no
    AND ec.birth_date >= '1970-01-01'
    AND ec.birth_date < '1975-01-01'
);
 first_name | last_name
------------+-----------
(0 rows)

EMP_s4903400=# 
```



```
EMP_s4903400=# EXPLAIN SELECT ep.first_name, ep.last_name
FROM employees_public ep
WHERE EXISTS (
    SELECT 1
    FROM employees_confidential_fdw ec
    WHERE ec.emp_no = ep.emp_no
    AND ec.birth_date >= '1970-01-01'
    AND ec.birth_date < '1975-01-01'
);
                                            QUERY PLAN
--------------------------------------------------------------------------------------------------
 Hash Semi Join  (cost=157.21..5993.20 rows=15 width=15)
   Hash Cond: (ep.emp_no = ec.emp_no)
   ->  Seq Scan on employees_public ep  (cost=0.00..5048.24 rows=300024 width=19)
   ->  Hash  (cost=157.03..157.03 rows=15 width=4)
         ->  Foreign Scan on employees_confidential_fdw ec  (cost=100.00..157.03 rows=15 width=4)
(5 rows)

EMP_s4903400=# 
```