Assignment 1: Distributed Databases

Task 1: Load Database

As this picture shown that I have successfully entered the PostgreSQL interactive terminal (psql). Now I can enter SQL commands.

```
s4951799@infs3200-f117632f:~$ psql
psql (17.4 (Ubuntu 17.4-1.pgdg24.04+2))
Type "help" for help.
```

Figure 1 enter psql

Create new database named “EMP_s4951799”. It clear that there is not a database named “EMP_s4951799” and I successfully create a new database in my database.

```
s4951799=# create database "EMP_s4951799";
CREATE DATABASE
```

Figure 2 create database “EMP_s4951799”

If there is a database named “EMP_s49519799” in my database, it will show that “database "EMP_s4951799" already exists”. Now you can drop the original database or continue the following step.

Connect database “EMP_s4951799”;

```
s4951799=# \c EMP_s4951799
You are now connected to database "EMP_s4951799" as user "s4951799".
```

Figure 3 connect database

The next step is that put the “Resource” folder in the left pane.

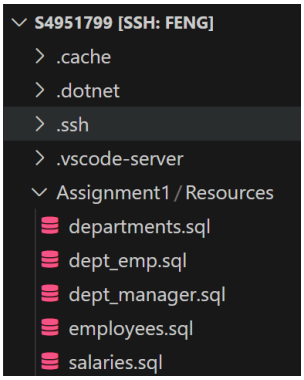


Figure 4 import folder

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

As required by task 1, I need execute the employees.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/employees.sql
CREATE TABLE
INSERT 0 10000
INSERT 0 10000
```

Figure 5 execute script 1.1

The table “employees” has existed in the “EMP_s4951799” database.

```
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 24
```

Figure 6 execute script 1.2

Now I can count the number of entries by using count function. As the picture shown that the number of the entries in the table “employs” is 300024.

```
EMP_s4951799=# select count(*) from employees;
 count
--------
 300024
(1 row)
```

Figure 7 count entries

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

From the conceptual model of ‘EMP’ database we can find that the “emp_no” and “dept_name” in two different tables. So there are two ways to resolve this task: 1. I can find the “Marketing” department’s dept_no firstly and then use the count function to get the number of employees.

Because this two tables have the same primary key “dept_no”. So I can use “left outer join” to connect two tables and use count function to get the number of employees.

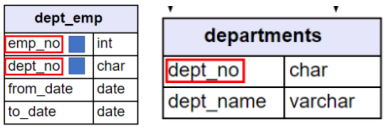


Figure 8 “dept_emp” and “departments” tables

Then I need to execute the departments.sql and dept_emp.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/departments.sql
CREATE TABLE
INSERT 0 9
```

Figure 9 execute script 2.1

```
EMP_s4951799=# \i ./Assignment1/Resources/dept_emp.sql
CREATE TABLE
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
```

Figure 10 execute script 2.2

```
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 1603
```

Figure 11 execute script 2.3

If I use the first solution to resolve this task, I need use two SQL queries to get the answer. First I find the “Marketing” dept_no is “d001” and then I use this dept_no to count the emp_no in the table “dept_emp”. Finally I get the answer that the number of employees is 20211.

```
EMP_s4951799=# select * from departments where dept_name='Marketing';
 dept_no | dept_name
---------+-----------
 d001    | Marketing
(1 row)
```

Figure 12 select dept_no of “Marketing” department

```
EMP_s4951799=# select count(emp_no) from dept_emp where dept_no='d001';
 count
-------
 20211
(1 row)
```

Figure 13 count the number of employees

If I use another way is to use outer join to connect two tables. By using the common primary key “dept_no”, I can connect the two tables. Finally I get the same answer that the number of employees is 20211.

```
EMP_s4951799=# select count(emp_no) from dept_emp left outer join departments on (dept_emp.
dept_no=departments.dept_no) where departments.dept_name='Marketing';
 count
-------
 20211
(1 row)
```

Figure 14 use left outer join command to count

Task 2: Database Fragmentation

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

Firstly execute the salaries.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/salaries.sql
CREATE TABLE
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
```

Figure 15 execute 3.1

```
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 4047
```

Figure 16 execute 3.2

As required by the task 2, I need perform horizontal fragmentation on table ‘salaries’ based on the ‘from_date’.

```
EMP_s4951799=# create table "salaries_h1" as select * from salaries where from_date < '1990
-01-01';
SELECT 286543
```

Figure 17 create table “salaries_h1”

```
EMP_s4951799=# create table "salaries_h2" as select * from salaries where from_date < '1992
-01-01' and from_date >= '1990-01-01';
SELECT 247185
```

Figure 18 create table “salaries_h2”

```
EMP_s4951799=# create table "salaries_h3" as
select * from salaries
where from_date < '1994-01-01' and from_date >= '1992-01-01';
SELECT 319211
```

Figure 19 create table “salaries_h3”

```
EMP_s4951799=# create table "salaries_h4" as
select * from salaries
where from_date < '1996-01-01' and from_date >= '1994-01-01';
SELECT 386796
```

Figure 20 create table “salaries_h4”

```
EMP_s4951799=# create table "salaries_h5" as
select * from salaries
where from_date < '1998-01-01' and from_date >= '1996-01-01';
SELECT 451499
```

Figure 21 create table “salaries_h5”

```
EMP_s4951799=# create table "salaries_h6" as
select * from salaries
where from_date < '2000-01-01' and from_date >= '1998-01-01';
SELECT 508446
```

Figure 22 create table “salaries_h6”

```
EMP_s4951799=# create table "salaries_h7" as
select * from salaries
where from_date >= '2000-01-01';
SELECT 644367
```

Figure 23 create table “salaries_h7”

Now I fragment the ‘salaries’ table into 7 tables based on ‘from_date’ and name them ‘salaries_h1’, ‘salaries_h2’, ‘salaries_h3’, ‘salaries_h4’, ‘salaries_h5’, ‘salaries_h6’, ‘salaries_h7’.

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.

From the requirement I know that the time between 1996-06-30 and 1996-12-31 locate in the table ‘salaries_h5’. So the next SQL queries need be commanded in the table ‘salaries_h5’ and I need use the average function to calculate average employee salary. And the answer of the average is 63693.220801456911.

```
EMP_s4951799=# select AVG(salary) from salaries_h5
where from_date <= '1996-12-31' and from_date >= '1996-06-01';
        avg
--------------------
 63693.220801456911
(1 row)
```

Figure 24 calculate the average of salary

If I want to get the explanation of this queries, I will use “explain analyze” command.

```
EMP_s4951799=# explain analyze
select AVG(salary) from salaries_h5
where from_date <= '1996-12-31' and from_date >= '1996-06-01';
```

Figure 25 analyze the SQL query

This figure displays the Query Plan of a PostgreSQL query, reflecting the steps, performance metrics, and resource usage of the database during the execution.

```
                                                                QUERY PLAN

-------------------------------------------------------------------------------------------
-----------------------------------------------
 Finalize Aggregate  (cost=7667.82..7667.83 rows=1 width=32) (actual time=27.311..44.581 ro
ws=1 loops=1)
   ->  Gather  (cost=7667.71..7667.82 rows=1 width=32) (actual time=27.158..44.574 rows=2 l
oops=1)
         Workers Planned: 1
         Workers Launched: 1
         ->  Partial Aggregate  (cost=6667.71..6667.72 rows=1 width=32) (actual time=20.724
..20.725 rows=1 loops=2)
               ->  Parallel Seq Scan on salaries_h5  (cost=0.00..6479.81 rows=75158 width=4
) (actual time=0.033..17.619 rows=64246 loops=2)
                     Filter: ((from_date <= '1996-12-31'::date) AND (from_date >= '1996-06-
01'::date))
                     Rows Removed by Filter: 161504
 Planning Time: 0.066 ms
 Execution Time: 44.606 ms
(10 rows)
```

Figure 26 query plan

This is the first logical step in the query execution. PostgreSQL applies the WHERE clause to filter rows from the salaries table. Only rows where the from_date is between '1996-06-01' and '1996-12-31' are retained.

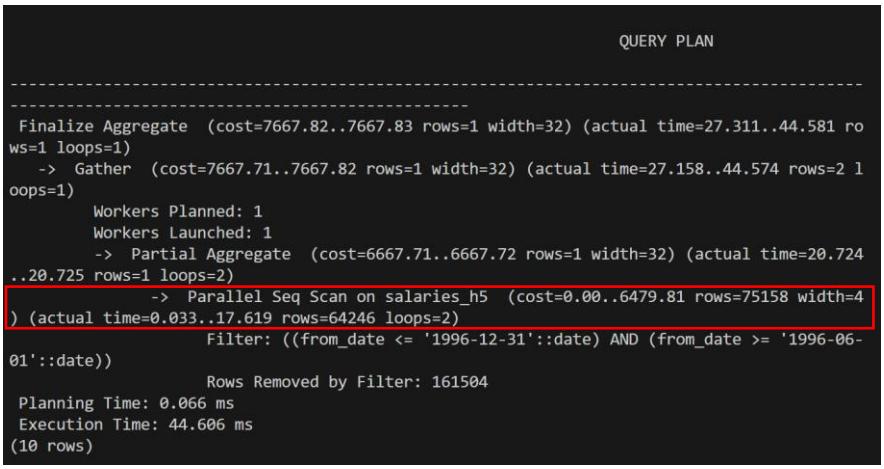


Figure 27 query plan2

PostgreSQL performs a Parallel Sequential Scan on the salaries_h5 table

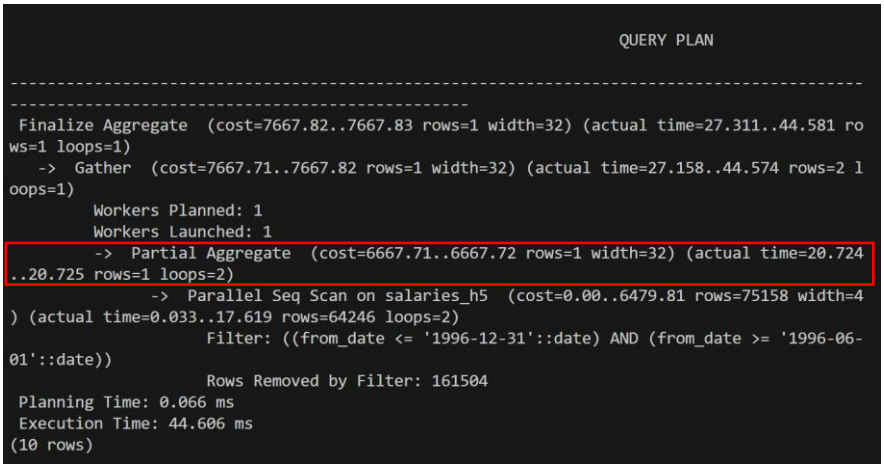


Figure 28 query plan3

This step computes a partial result of the aggregation

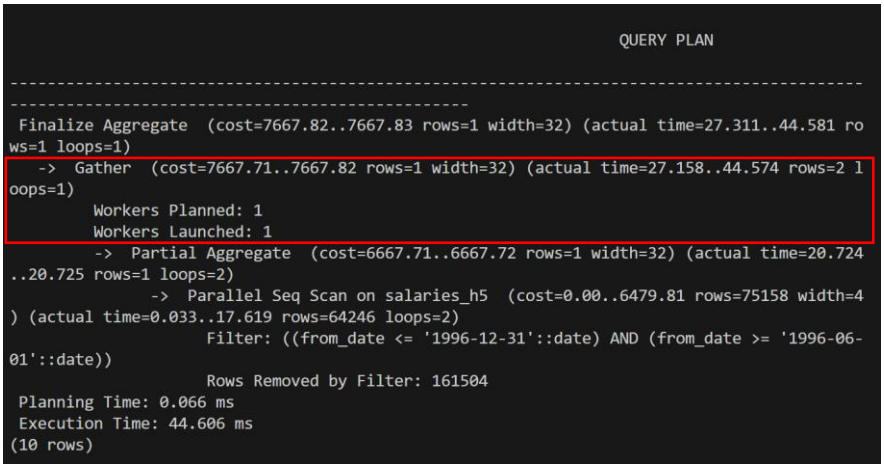


Figure 29 query plan4

The Gather node collects the partial results from the parallel workers and combines them into a single result.

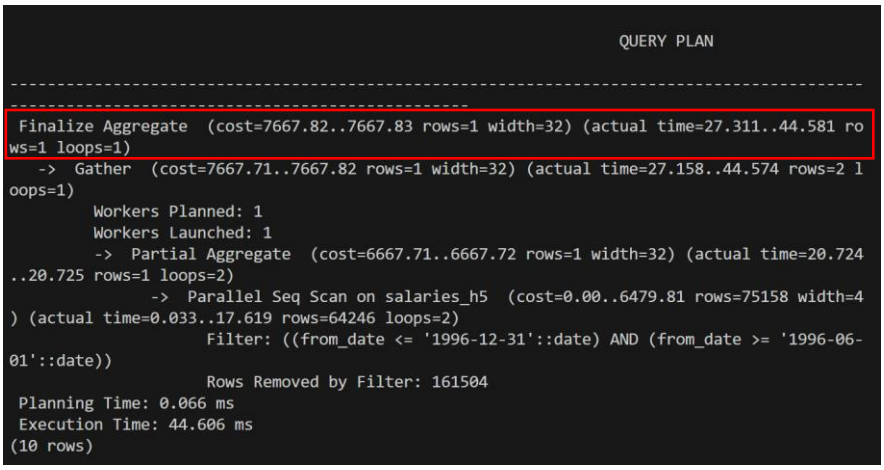


Figure 30 query plan5

PostgreSQL completes the aggregation by finalizing the result.

How PSQL database system optimizesthe queries?

1. PostgreSQL could reduce the overall execution time by splitting the work across multiple workers.
2. The filter on from_date is applied as early as possible to reduce the number of rows.

Write sql queries to perform vertical fragmentation on table 'employee'

Create “employees_public” table and insert data into this table in the new SQL file.

```
EMP_s4951799=# CREATE TABLE "employees_public" (
    emp_no INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE
);

INSERT INTO employees_public (emp_no, first_name, last_name, hire_date)
SELECT emp_no, first_name, last_name, hire_date FROM employees;
CREATE TABLE
INSERT 0 300024
```

Figure 31 create table “employees_public”

Create new table named “employees_confidential” and insert data into new table in the new SQL file.

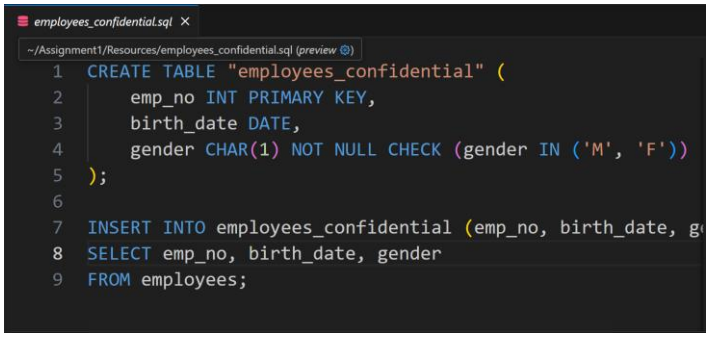


Figure 32 create table “employees_confidential”

Execute employees_confidential.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/employees_confidential.sql
CREATE TABLE
INSERT 0 300024
```

Figure 33 execute script

Run the pg_dump command. And export table emp_confidential to a SQL file.

```
s4951799@infs3200-f117632f:~$ pg_dump -U s4951799 -d EMP_s4951799 -t employees_confid
ential > employe
es_confidential.sql
```

Figure 34 Run the following command from the terminal

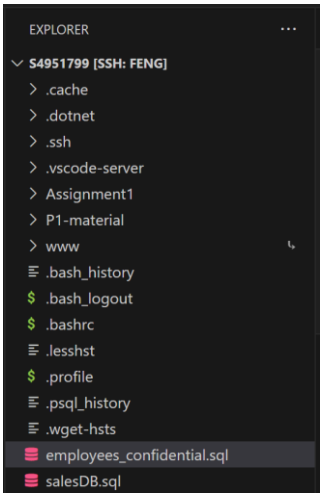


Figure 35 employees_confidential.sql file

Create new database named “EMP_confidential” and connect the new database.

```
EMP_s4951799=# create database "EMP_confidential";
CREATE DATABASE
EMP_s4951799=# \c "EMP_confidential";
You are now connected to database "EMP_confidential" as user "s4951799".
```

Figure 36 create and connect database “EMP_confidential”

Execute employees_confidential.sql script in database EMP_confidential.

```
EMP_confidential=# \i ./employees_confidential.sql
SET
SET
SET
SET
SET
SET
 set_config
```

Figure 37 execute script 4.1

```
SET
SET
SET
CREATE TABLE
ALTER TABLE
COPY 300024
ALTER TABLE
```

Figure 38 execute script 4.2

Exit the psql shell.

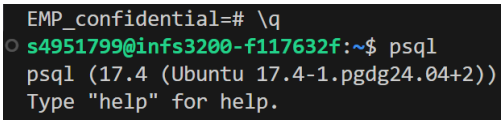


Figure 39 Exit psql

Connect database EMP_confidential and list all tables in the current database and schema. As this picture showed that there is only one table named "employees_confidential" in the "EMP_confiential" database.

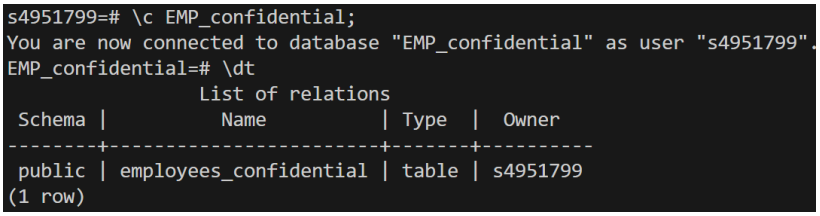


Figure 40 list all tables from database

Task 3: Database Replication

Consider you have 5 different server sites. How to design full replication/ partial replication/ no replication? What could be the pros and cons for each fragmentation strategy?

Firstly I create 5 server sites and name them respectively.

- site1_s4951799
- site2_ s4951799
- site3_ s4951799
- site4_ s4951799
- site5_ s4951799

1. full replication
   In these five server sites, each site has a full copy of each fragment. I should create five sites to simulate the full replication in psql command line.
   Pros: every site can resolve independently handle all queries.
   Cons: There are increasing storage requirements if the data volume is large. And when updating a record, all copies across all sites must be synchronized.
2. partial replication
   In these five server sites, more than one site may have a copy of this fragment, but not all of them.
   • site1_ s4951799, which contains:
   Employees_PA1
   Employees_PA2
   Employees_PA3
   • site2_ s4951799, which contains:
   Employees_PA2
   Employees_PA3
   Employees_PA4
   • site3_ s4951799, which contains:
   Employees_PA4
   Employees_PA5
   Employees_PA6
   • site4_ s4951799, which contains:
   Employees_PA6
   Employees_PA7
   Employees_PA8
   • site5_ s4951799, which contains:
   Employees_PA8
   Employees_PA9
   Employees_PA10
   Pros: Partial replication has high fault tolerance. If a site goes down, other sites may still have the required fragment.
   Cons: If a site needs data that is not stored locally, it must access another site.

3. no replication
   Each fragment will be a relation located on only one site.
   • site1_ s4951799, which contains:
     Employees_PA1
     Employees_PA2
   • site2_ s4951799, which contains:
     Employees_PA3
     Employees_PA4
   • site3_ s4951799, which contains:
     Employees_PA5
     Employees_PA6
   • site4_ s4951799, which contains:
     Employees_PA7
     Employees_PA8
   • site5_ s4951799, which contains:
     Employees_PA9
     Employees_PA10
   Pros: Each fragment just has one copy, minimizing the whole storage requirements. And updates only need to modify the data on a single site, with no synchronization required.
   Cons: The no replication has low fault tolerance. If a site goes down, the fragments it holds become unavailable.

Consider your master server has the fragmentation allocation schema. Based on your design, what is the process to update a record with a specific 'emp_no'?

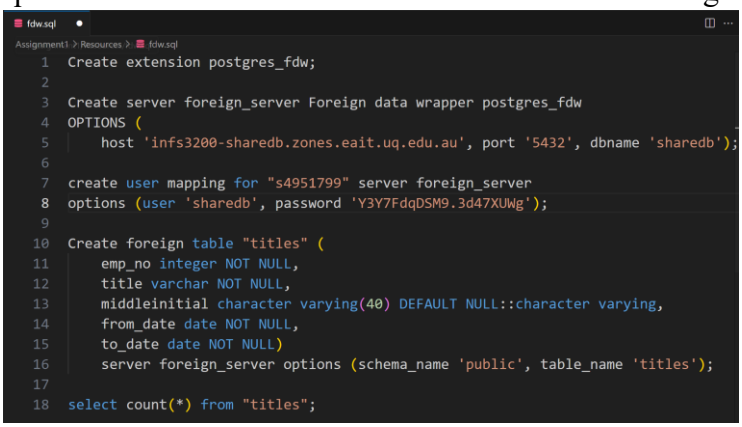
If I want to update a record, such as the employee information for “ emp_no=’10006’ ”.

1. full replication
   I need update this record on each site, because every site has this information.
2. partial replication
   The master server determines which fragment contains emp_no=’10006’ based on the allocation schema. Assuming the emp_no=’10006’ belongs to Employees_PA2. The master server checks the allocation schema and finds that Employees_PA2 is stored on site1_ s4951799 and site2_ s4951799. The master server sends the update request to site1 and site2. Site1 and site2 receive the request, locate the record for emp_no=’10006’ in its local Employees_PA2, and applies the update.
3. no replication
   The master server determines which fragment contains emp_no=’10006’ based on the allocation schema. Assuming the emp_no=’10006’ belongs to Employees_PA2. The master server checks the allocation schema and finds that Employees_PA2 is stored on site1_ s4951799. The master server sends the update request to site1. Site1 receives the request, locates the record for emp_no=’10006’ in its local Employees_PA2, and applies the update.

Task 4: Access foreign data with FDW

Write sql queries to establish a FDW to the external DB. Queries/commands should be provided.

Create a new sql file named “fdw”. Access the remote database through FDW.



```sql
Create extension postgres_fdw;

Create server foreign_server Foreign data wrapper postgres_fdw
OPTIONS (
    host 'infs3200-sharedb.zones.eait.uq.edu.au', port '5432', dbname 'sharedb');

create user mapping for "s4951799" server foreign_server
options (user 'sharedb', password 'Y3Y7FdqDSM9.3d47XUWg');

Create foreign table "titles" (
    emp_no integer NOT NULL,
    title varchar NOT NULL,
    middleinitial character varying(40) DEFAULT NULL::character varying,
    from_date date NOT NULL,
    to_date date NOT NULL)
    server foreign_server options (schema_name 'public', table_name 'titles');

select count(*) from "titles";
```

Figure 41 create fdw.sql

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

1. Execute the fdw.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/fdw.sql
CREATE EXTENSION
CREATE SERVER
CREATE USER MAPPING
CREATE FOREIGN TABLE
 count
--------
 443308
(1 row)
```

Figure 42 execute script

2. Calculate the salary of the different titles by using AVG function.

```
EMP_s4951799=# SELECT t.title, AVG(s.salary) AS avg_salary
FROM titles 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_salary
--------------------+--------------------
 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)
```

Figure 43 calculate average

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.

Creates a new user named emp_user with the password ‘infs3200’. Allows emp_user to connect to the database EMP_s4951799, emp_user to use tables inside the public schema and emp_user to read data from the employees_public table.

```
EMP_s4951799=# create user emp_user with password 'infs3200';
CREATE ROLE
EMP_s4951799=# grant connect on database "EMP_s4951799" to emp_user;
GRANT
EMP_s4951799=# grant usage on schema public to emp_user;
GRANT
EMP_s4951799=# grant select on table employees_public to emp_user;
GRANT
```

Figure 44 create user

Connect database EMP_confidential.

```
EMP_s4951799=# \c EMP_confidential;
You are now connected to database "EMP_confidential" as user "s4951799".
```

Figure 45 connect database EMP_confidential

Allows emp_user to connect to the database EMP_confidential, emp_user to use tables inside the public schema and emp_user to read data from the employees_confidential table.

```
EMP_confidential=# grant connect on database "EMP_confidential" to emp_user;
GRANT
EMP_confidential=# grant usage on schema public to emp_user;
GRANT
EMP_confidential=# grant select on table employees_confidential to emp_user;
GRANT
```

Figure 46 connect user

Create a new sql file named “confidential_fdw”.

confidential_fdw.sql | employees_confidential.sql | v2_fdw.sql

Assignment1 > Resources > confidential_fdw.sql

```
1  CREATE SERVER remote_dates_server
2  FOREIGN DATA WRAPPER postgres_fdw
3  OPTIONS (host 'localhost', port '5432', dbname 'EMP_confidential');
4
5  CREATE USER MAPPING FOR "s4951799"
6  SERVER remote_dates_server
7  OPTIONS (user 'emp_user', password 'infs3200');
8
9  CREATE FOREIGN TABLE emp_dates (
10     emp_no INT,
11     birth_date DATE,
12     gender CHAR(1) NOT NULL CHECK (gender IN ('M', 'F'))
13 )
14 SERVER remote_dates_server
15 OPTIONS (schema_name 'public', table_name 'employees_confidential');
```

Figure 47 create confidential_fdw.sql

Execute the confidential_fdw.sql script.

```
EMP_s4951799=# \i ./Assignment1/Resources/confidential_fdw.sql
CREATE SERVER
CREATE USER MAPPING
CREATE FOREIGN TABLE
```

Figure 48 execute script

Query using the semi-join method. Firstly transmit foreign keys/primary keys from local site to remote site to prepare semi join.

```
EMP_s4951799=# select emp_no from employees_public;
```

Figure 49 semi-join 1

The employee number in the range where the query date is executed in the remote table

```
EMP_s4951799=# select emp_dates.emp_no, emp_dates.birth_date from emp_dates, (select emp_no from employees_pub
lic) FS where emp_dates.emp_no=FS.emp_no and birth_date >= '1970-01-01'
  AND birth_date < '1975-01-01';
 emp_no | birth_date
--------+------------
(0 rows)
```

Figure 50 semi-join 2

Returns the query result to the local table.

```
EMP_s4951799=# select FN.emp_no, FN.birth_date, employees_public.first_name, employees_public.last_name from e
mployees_public, (select emp_dates.emp_no, emp_dates.birth_date from emp_dates, (select emp_no from employees_
public) FS where emp_dates.emp_no=FS.emp_no and birth_date >= '1970-01-01'
  AND birth_date < '1975-01-01') FN where employees_public.emp_no=FN.emp_no;
 emp_no | birth_date | first_name | last_name
--------+------------+------------+-----------
(0 rows)
```

Figure 51 semi-join 3

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

Analyze the semi-join:

It can be seen from the figure that the transmission cost of the first step of semi-join is 4 × 300024=1200096.

```
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------
----------
 Seq Scan on employees_public  (cost=0.00..5008.24 rows=300024 width=4) (actual time=0.010..19.412 rows=300024
 loops=1)
 Planning Time: 0.048 ms
 Execution Time: 29.934 ms
(3 rows)
```

Figure 52 analyze semi-join1

The transmission cost of the next step of the semi-join is 8×0=0.

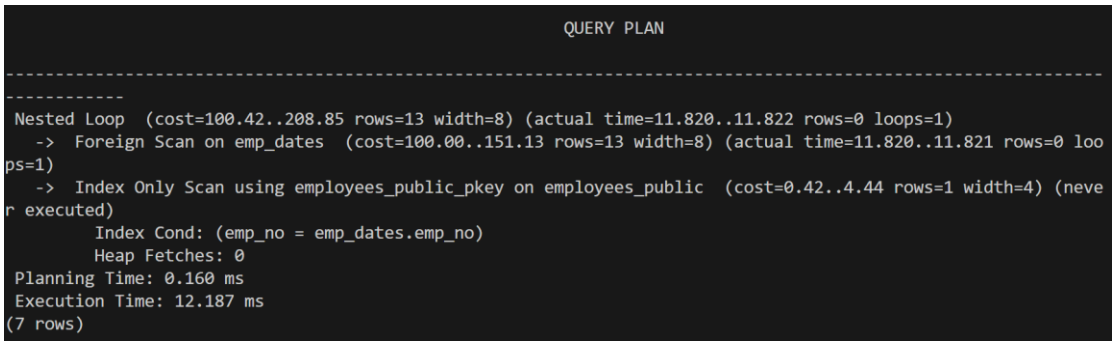


Figure 53 analyze semi-join2

The transmission cost of the third step of the semi-join is 23×0=0.

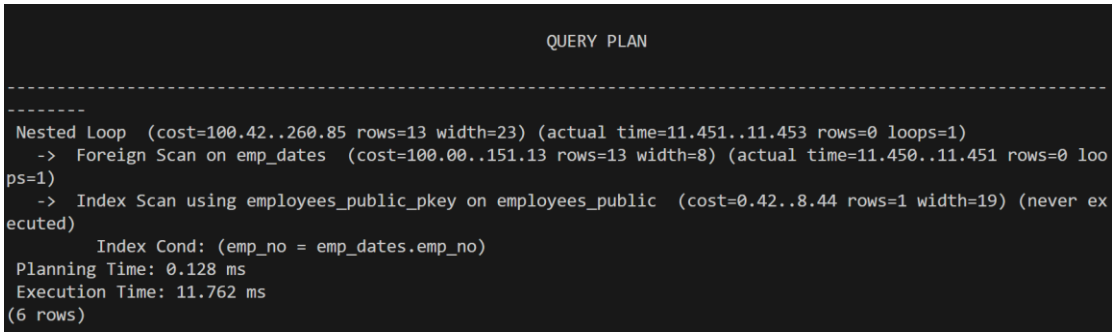


Figure 54 analyze semi-join3

The transmission cost of the total steps of the semi-join is 1200096 + 0 + 0=1200096.

Query using the inner-join method. Join the two tables first.

```
EMP_s4951799=# select emp_dates.emp_no, emp_dates.birth_date, employees_public.first_name, employees_public.last_name from employees_public, emp_dates where employees_public.emp_no=emp_dates.emp_no;
```

Figure 55 inner-join1

Complete the inner-join with conditions.

```
EMP_s4951799=# select FS.emp_no, FS.birth_date, FS.first_name, FS.last_name from (select emp_dates.emp_no, emp_dates.birth_date, employees_public.first_name, employees_public.last_name from employees_public, emp_dates where employees_public.emp_no=emp_dates.emp_no) FS where birth_date >= '1970-01-01' AND birth_date < '1975-01-01';
 emp_no | birth_date | first_name | last_name
--------+------------+------------+-----------
(0 rows)
```

Figure 56 inner-join2

Analyze the inner-join:

It can be seen from the figure that the transmission cost of the first step of inner-join is 23×300024=6900552.

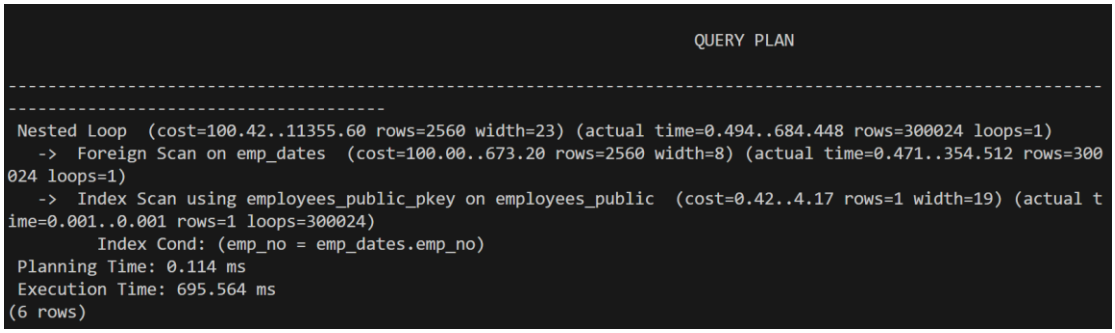


Figure 57analyze inner-join1

The transmission cost of the second step of inner-join is 23×0=0.

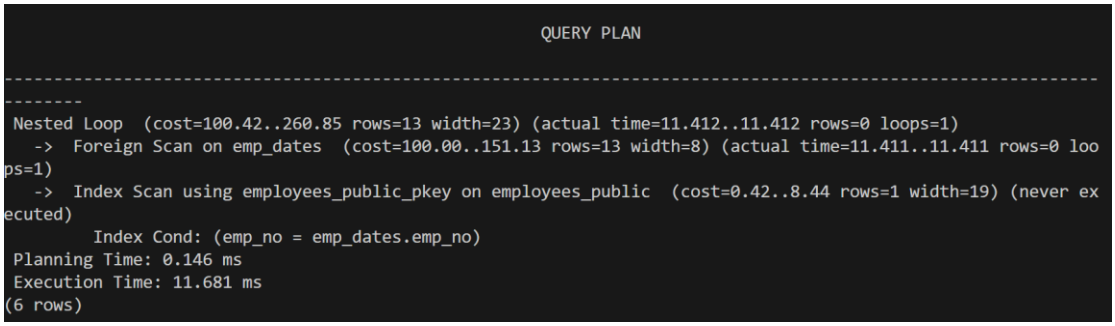


Figure 58 analyze inner-join2

The transmission cost of the total steps of the inner-join is 6900552 + 0=6900552.

The transmission cost of semi-join is 1200096 bytes, which is smaller than that of inner-join (6900552 bytes).The difference of transmission cost between internal connection and semi-connection is mainly due to the timing and transmission volume of data screening: the semi-connection filters the remote table emp_dates before transmission, and transmits only the conditional rows, which reduces the cost with high selectivity. The inner connection directly transmits the entire local surface emp_public and returns the result, not optimizing the transfer volume, resulting in a cost of up to 6900552 bytes. Therefore, in highly selective scenarios, the transmission cost of semi-connection is much lower than that of internal connection.