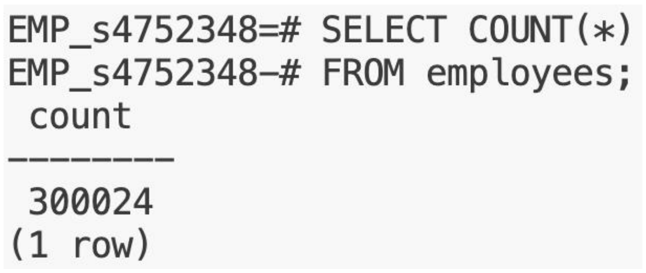
INFS3200 Advanced Database System Assignment 1

Student Name: Wei-Ting, Hong

Student Number: 47523483

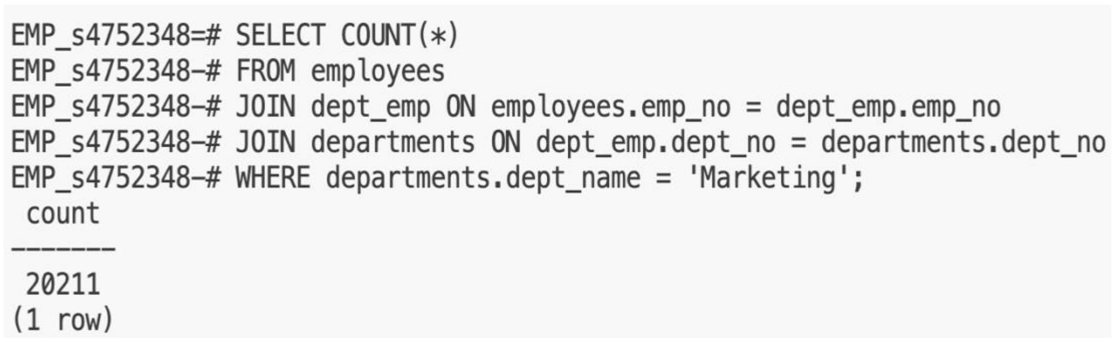
Task 1.

(1).



```
EMP_s4752348=# SELECT COUNT(*)
EMP_s4752348-# FROM employees;
 count
--------
 300024
(1 row)
```

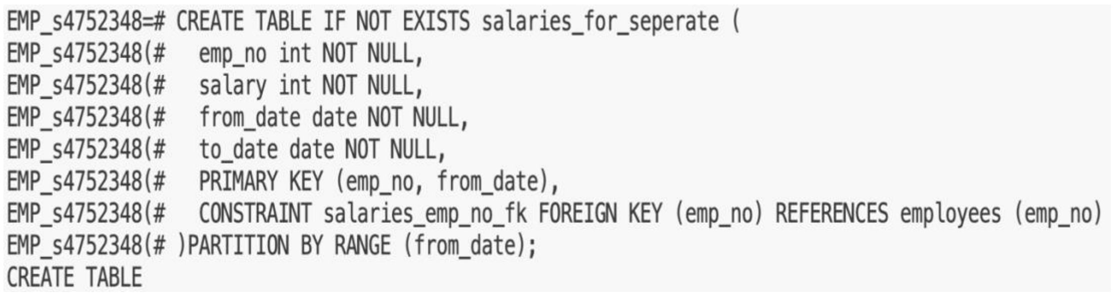
(2).



```
EMP_s4752348=# SELECT COUNT(*)
EMP_s4752348-# FROM employees
EMP_s4752348-# JOIN dept_emp ON employees.emp_no = dept_emp.emp_no
EMP_s4752348-# JOIN departments ON dept_emp.dept_no = departments.dept_no
EMP_s4752348-# WHERE departments.dept_name = 'Marketing';
 count
-------
 20211
(1 row)
```

Task 2.

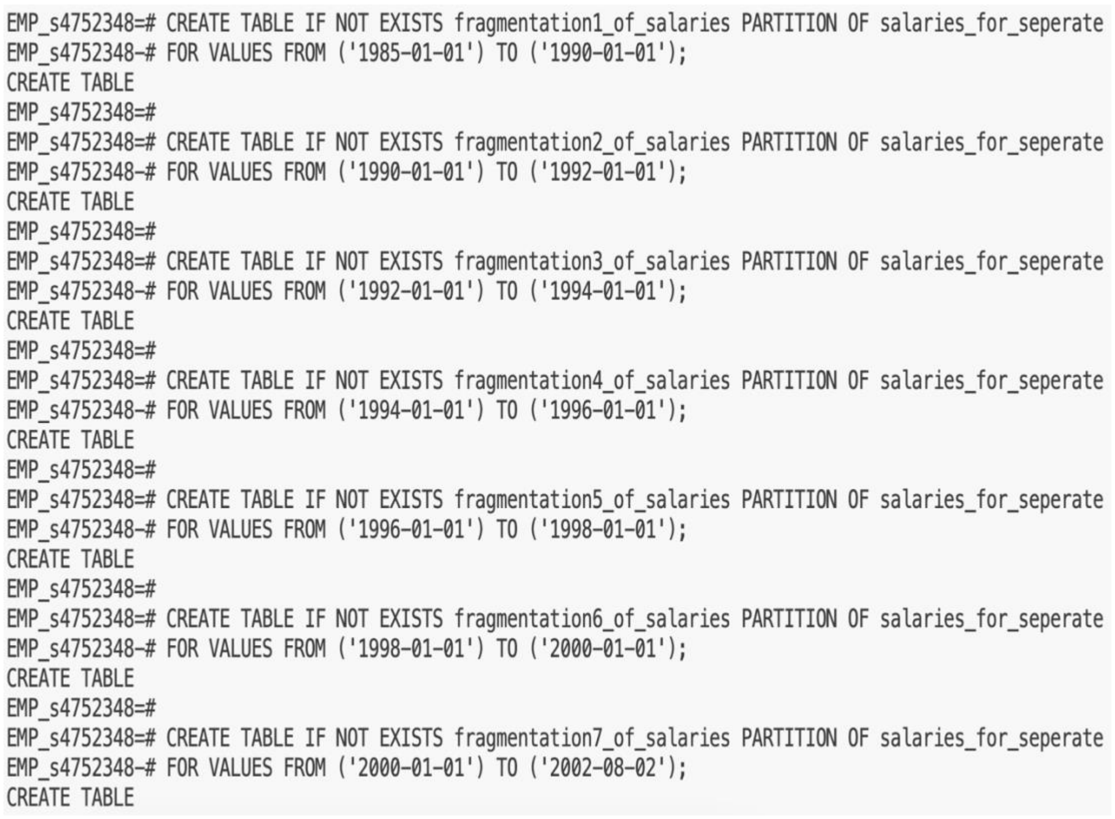
Before answering the questions, we first need to create a new table with clear definition on how to separate the fragmentations, in this case, we use “from_date” as the separation standard. Therefore, he last sentence of the SQL should be `PARTITION BY RANGE (from_date);`, then we could start the separation.



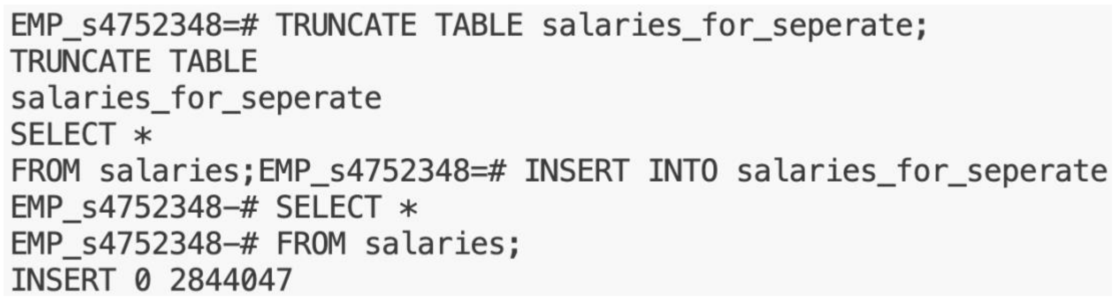
```
EMP_s4752348=# CREATE TABLE IF NOT EXISTS salaries_for_seperate (
EMP_s4752348(#   emp_no int NOT NULL,
EMP_s4752348(#   salary int NOT NULL,
EMP_s4752348(#   from_date date NOT NULL,
EMP_s4752348(#   to_date date NOT NULL,
EMP_s4752348(#   PRIMARY KEY (emp_no, from_date),
EMP_s4752348(#   CONSTRAINT salaries_emp_no_fk FOREIGN KEY (emp_no) REFERENCES employees (emp_no)
EMP_s4752348(# )PARTITION BY RANGE (from_date);
CREATE TABLE
```

(1) For fragmentation of 'from_date' before 1990-01-01, first we could use

SELECT clause to obtain the earliest "from_date" in table "employees", which was "1985-01-01". In terms of fragmentation of 'from_date' no earlier than 2000-01-01, the latest "from_date" in table "employees" was "2002-08-02". After confirming the earliest and latest "from_date", we could separate all required fragmentations as follow:

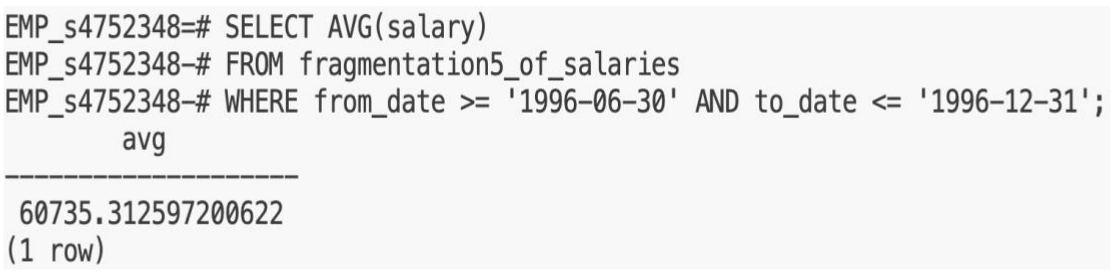


```
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation1_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1985-01-01') TO ('1990-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation2_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1990-01-01') TO ('1992-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation3_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1992-01-01') TO ('1994-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation4_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1994-01-01') TO ('1996-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation5_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1996-01-01') TO ('1998-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation6_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('1998-01-01') TO ('2000-01-01');
CREATE TABLE
EMP_s4752348=#
EMP_s4752348=# CREATE TABLE IF NOT EXISTS fragmentation7_of_salaries PARTITION OF salaries_for_seperate
EMP_s4752348-# FOR VALUES FROM ('2000-01-01') TO ('2002-08-02');
CREATE TABLE
```

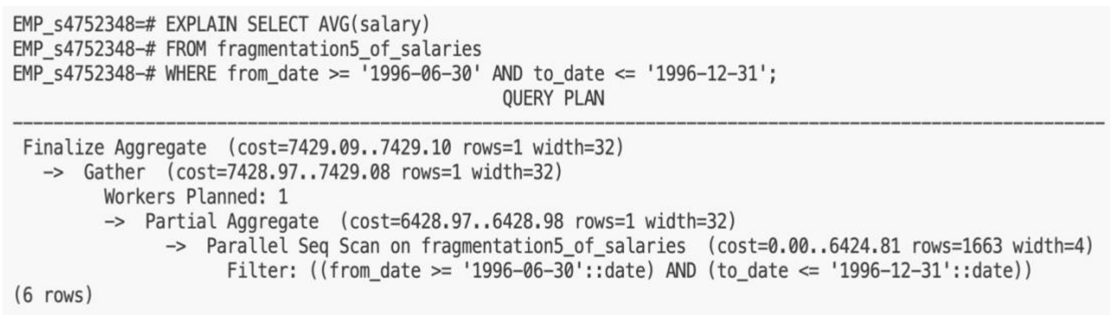


```
EMP_s4752348=# TRUNCATE TABLE salaries_for_seperate;
TRUNCATE TABLE
salaries_for_seperate
SELECT *
FROM salaries;EMP_s4752348=# INSERT INTO salaries_for_seperate
EMP_s4752348-# SELECT *
EMP_s4752348-# FROM salaries;
INSERT 0 2844047
```

(2) The query screenshot and explanation are shown as follow:



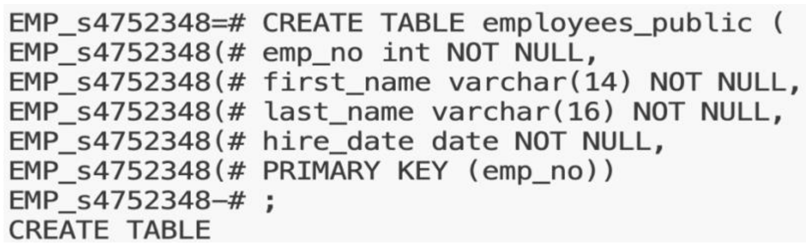
```
EMP_s4752348=# SELECT AVG(salary)
EMP_s4752348-# FROM fragmentation5_of_salaries
EMP_s4752348-# WHERE from_date >= '1996-06-30' AND to_date <= '1996-12-31';
        avg
--------------------
 60735.312597200622
(1 row)
```



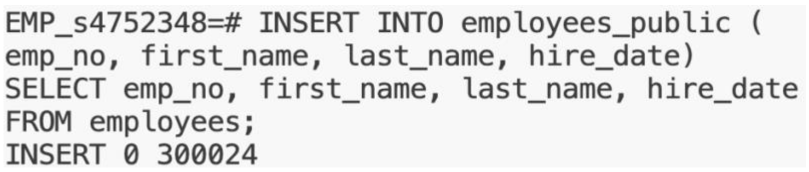
```
EMP_s4752348=# EXPLAIN SELECT AVG(salary)
EMP_s4752348-# FROM fragmentation5_of_salaries
EMP_s4752348-# WHERE from_date >= '1996-06-30' AND to_date <= '1996-12-31';
                                               QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=7429.09..7429.10 rows=1 width=32)
   ->  Gather  (cost=7428.97..7429.08 rows=1 width=32)
         Workers Planned: 1
         ->  Partial Aggregate  (cost=6428.97..6428.98 rows=1 width=32)
               ->  Parallel Seq Scan on fragmentation5_of_salaries  (cost=0.00..6424.81 rows=1663 width=4)
                     Filter: ((from_date >= '1996-06-30'::date) AND (to_date <= '1996-12-31'::date))
(6 rows)
```

First, the database system scans all records in the target file and leaves only records that match the WHERE clause. Second, the database system will perform partial aggregate. It will divide the records which need to be calculated by the aggregation function into several small pieces, and then perform gather operation after calculating them separately. By collecting the answers from the partial calculations, finally, the result is calculated and return.

(3) The first two screen is to create first vertical table “employees_public” in original database.

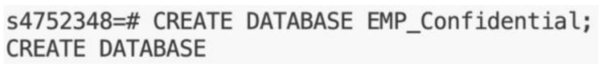


```
EMP_s4752348=# CREATE TABLE employees_public (
EMP_s4752348(# emp_no int NOT NULL,
EMP_s4752348(# first_name varchar(14) NOT NULL,
EMP_s4752348(# last_name varchar(16) NOT NULL,
EMP_s4752348(# hire_date date NOT NULL,
EMP_s4752348(# PRIMARY KEY (emp_no))
EMP_s4752348-# ;
CREATE TABLE
```



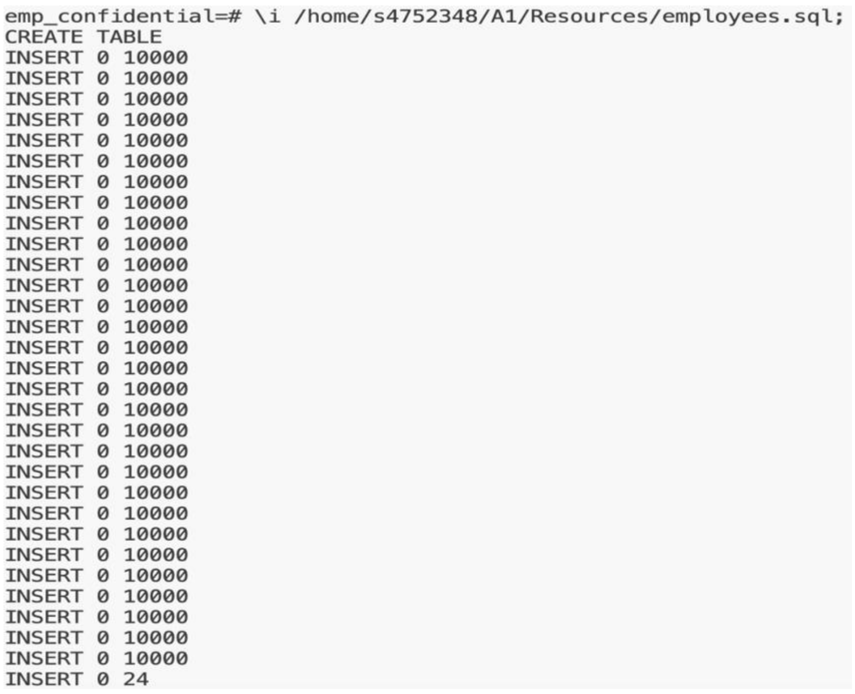
```
EMP_s4752348=# INSERT INTO employees_public (
emp_no, first_name, last_name, hire_date)
SELECT emp_no, first_name, last_name, hire_date
FROM employees;
INSERT 0 300024
```

For the second vertical fragmentation, it should be stored in new DB “EMP_Confidential”, so the first step is to create the new DB.



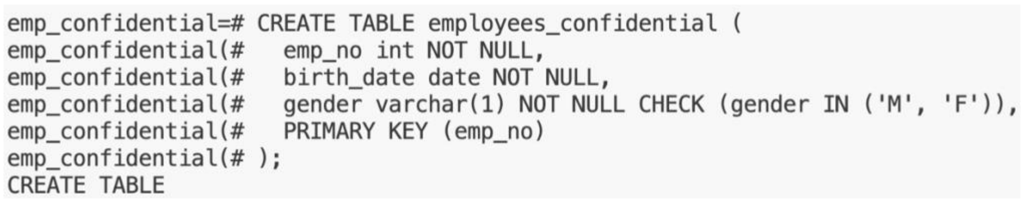
```
s4752348=# CREATE DATABASE EMP_Confidential;
CREATE DATABASE
```

Then we import the master table “employees” into the new DB.

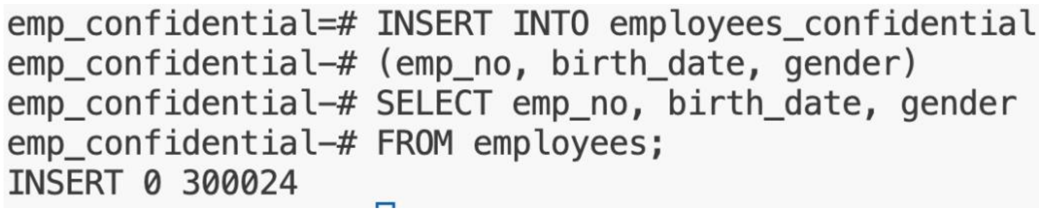


```
emp_confidential=# \i /home/s4752348/A1/Resources/employees.sql;
CREATE TABLE
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 10000
INSERT 0 24
```

Next step is to define the vertical fragmentation and import the qualified data into it.



```
emp_confidential=# CREATE TABLE employees_confidential (
emp_confidential(#   emp_no int NOT NULL,
emp_confidential(#   birth_date date NOT NULL,
emp_confidential(#   gender varchar(1) NOT NULL CHECK (gender IN ('M', 'F')),
emp_confidential(#   PRIMARY KEY (emp_no)
emp_confidential(# );
CREATE TABLE
```



```
emp_confidential=# INSERT INTO employees_confidential
emp_confidential-# (emp_no, birth_date, gender)
emp_confidential-# SELECT emp_no, birth_date, gender
emp_confidential-# FROM employees;
INSERT 0 300024
```

Task (3).

• Full replication method stores every fragmentation in every site, i.e., each site would have ten fragmentations. The advantages of this method would be the high accessibility, every site has whole data, we could access every data we need in any site. Even though some of the sites is not working, this issue would not influence the operation at all. The disadvantage is that the storage cost, we store entire table data in every site, we may need to prepare bigger site capacity. In addition, update processes may be significantly expensive as well. The reason is every site should always store identical data, so if we update one of the fragmentations in single site, the other sites should be updated as well, it would require considerable cost.

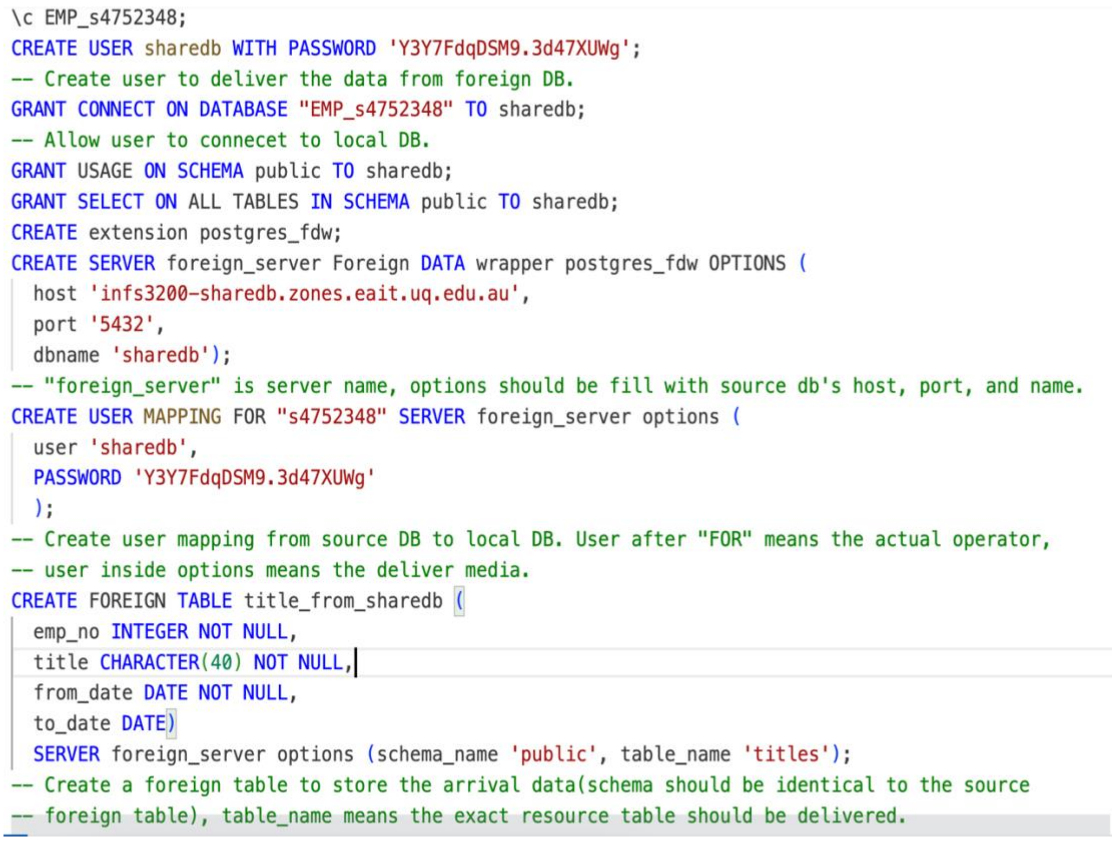
Partial replication method makes every site store some of the fragmentations but not all. And the overlapping store may occur. The advantage of this method is it could save the storage cost, each site only needs to store the distributed data pieces. This condition could also increase loading efficiency, we just only need to take involved fragmentation into account while querying. The disadvantage would be the complicated query processes and combination of query results. Scattered data would require extra actions to combine for obtaining the final result.

No replication method also makes every site to store some fragmentations, however, the overlapping would not occur. The pros and cons of this method basically are more extreme than partial replication method. Moreover, if one of the site is not working, than the query would not be approved to execute because no overlapping is allowed in this method.

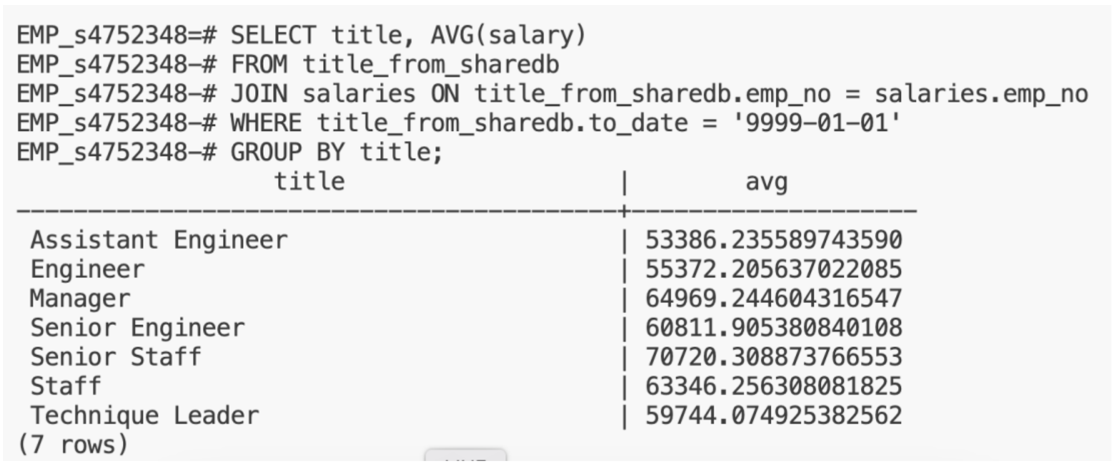
• As industry convention, I will choose partial replication to manage the distributed database. For updating a specific record, first I will check which fragmentations contain the record is going to update by using primary key. Next, I will start to execute queries on these selected sites, to ensure the consistency, I will use 2PC or even 3PC methods. After the queries are successfully executed, the final result should return.

Task (4).

(1)(2).

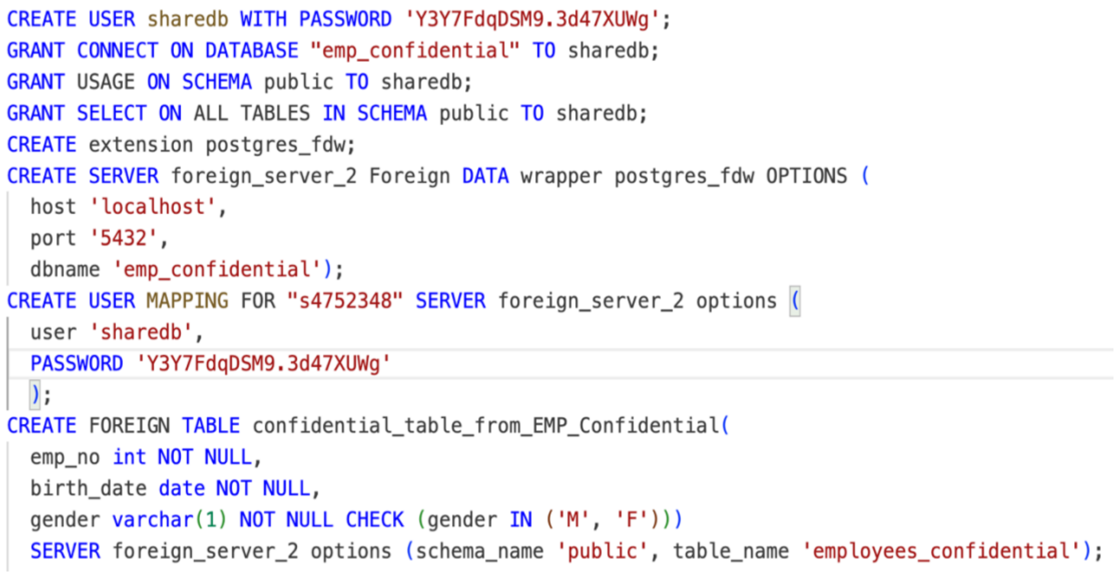


```
\c EMP_s4752348;
CREATE USER sharedb WITH PASSWORD 'Y3Y7FdqDSM9.3d47XUWg';
-- Create user to deliver the data from foreign DB.
GRANT CONNECT ON DATABASE "EMP_s4752348" TO sharedb;
-- Allow user to connecet to local DB.
GRANT USAGE ON SCHEMA public TO sharedb;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO sharedb;
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');
-- "foreign_server" is server name, options should be fill with source db's host, port, and name.
CREATE USER MAPPING FOR "s4752348" SERVER foreign_server options (
  user 'sharedb',
  PASSWORD 'Y3Y7FdqDSM9.3d47XUWg'
  );
-- Create user mapping from source DB to local DB. User after "FOR" means the actual operator,
-- user inside options means the deliver media.
CREATE FOREIGN TABLE title_from_sharedb (
  emp_no INTEGER NOT NULL,
  title CHARACTER(40) NOT NULL,
  from_date DATE NOT NULL,
  to_date DATE)
  SERVER foreign_server options (schema_name 'public', table_name 'titles');
-- Create a foreign table to store the arrival data(schema should be identical to the source
-- foreign table), table_name means the exact resource table should be delivered.
```



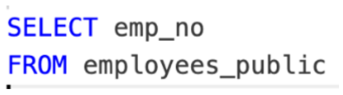
```
EMP_s4752348=# SELECT title, AVG(salary)
EMP_s4752348-# FROM title_from_sharedb
EMP_s4752348-# JOIN salaries ON title_from_sharedb.emp_no = salaries.emp_no
EMP_s4752348-# WHERE title_from_sharedb.to_date = '9999-01-01'
EMP_s4752348-# GROUP BY title;
                  title                  |         avg
-----------------------------------------+--------------------
 Assistant Engineer                      | 53386.235589743590
 Engineer                                | 55372.205637022085
 Manager                                 | 64969.244604316547
 Senior Engineer                         | 60811.905380840108
 Senior Staff                            | 70720.308873766553
 Staff                                   | 63346.256308081825
 Technique Leader                        | 59744.074925382562
(7 rows)
```

(3) Basically, we do the same thing as (1) for building a new foreign table, with several updates (name of source table, store table, server name, change host to localhost because both of source table and store table are in local database warehouse) of SQL to make the correct FDW.



```
CREATE USER sharedb WITH PASSWORD 'Y3Y7FdqDSM9.3d47XUWg';
GRANT CONNECT ON DATABASE "emp_confidential" TO sharedb;
GRANT USAGE ON SCHEMA public TO sharedb;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO sharedb;
CREATE extension postgres_fdw;
CREATE SERVER foreign_server_2 Foreign DATA wrapper postgres_fdw OPTIONS (
  host 'localhost',
  port '5432',
  dbname 'emp_confidential');
CREATE USER MAPPING FOR "s4752348" SERVER foreign_server_2 options (
  user 'sharedb',
  PASSWORD 'Y3Y7FdqDSM9.3d47XUWg'
  );
CREATE FOREIGN TABLE confidential_table_from_EMP_Confidential(
  emp_no int NOT NULL,
  birth_date date NOT NULL,
  gender varchar(1) NOT NULL CHECK (gender IN ('M', 'F')))
  SERVER foreign_server_2 options (schema_name 'public', table_name 'employees_confidential');
```

For implement the semi join, first we need to SELECT join key for preparation from local table. Like screen below.



```
SELECT emp_no
FROM employees_public
```

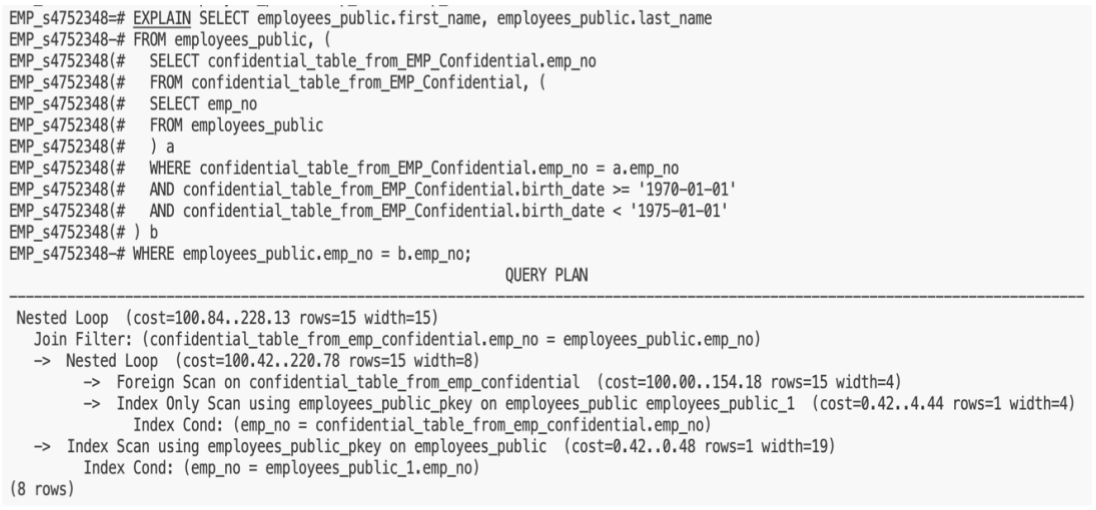
Next, we perform a query to SELECT the foreign/primary key from foreign table. The table inside FROM clause is foreign table, and another one is all the primary keys from local one. This operation is like JOIN logic in semi join, however, it didn't join all of the attributes to local table. We only need it to return the keys that satisfied the WHEHE clause.

```sql
SELECT confidential_table_from_EMP_Confidential.emp_no
FROM confidential_table_from_EMP_Confidential, (
  SELECT emp_no
  FROM employees_public
) a
WHERE confidential_table_from_EMP_Confidential.emp_no = a.emp_no
AND confidential_table_from_EMP_Confidential.birth_date >= '1970-01-01'
AND confidential_table_from_EMP_Confidential.birth_date < '1975-01-01';
```

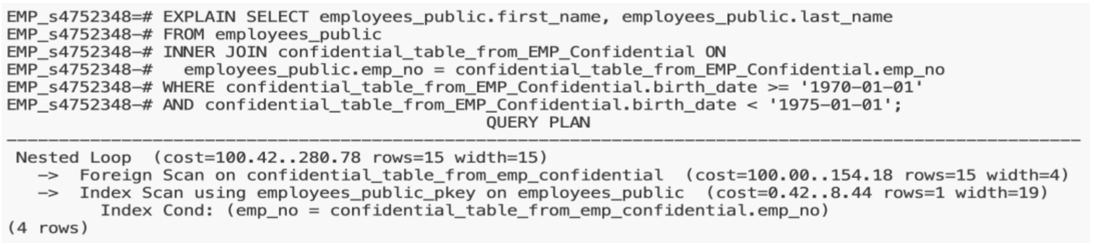
Now, we obtain the emp_no that birth date is no earlier than '1970-01-01'and before '1975-01-01'. The final step is that SELECT the final attributes needed from local table and those records' keys which match the WHERE clause (which is check for birth date) in step two. WHERE clause is another logic like JOIN in semi join.

```sql
SELECT employees_public.first_name, employees_public.last_name
FROM employees_public, (
  SELECT confidential_table_from_EMP_Confidential.emp_no
  FROM confidential_table_from_EMP_Confidential, (
  SELECT emp_no
  FROM employees_public
  ) a
  WHERE confidential_table_from_EMP_Confidential.emp_no = a.emp_no
  AND confidential_table_from_EMP_Confidential.birth_date >= '1970-01-01'
  AND confidential_table_from_EMP_Confidential.birth_date < '1975-01-01'
) b
WHERE employees_public.emp_no = b.emp_no;
```

(4) The following two screenshots are explanations of using semi and inner join:



```
EMP_s4752348=# EXPLAIN SELECT employees_public.first_name, employees_public.last_name
EMP_s4752348-# FROM employees_public, (
EMP_s4752348(#   SELECT confidential_table_from_EMP_Confidential.emp_no
EMP_s4752348(#   FROM confidential_table_from_EMP_Confidential, (
EMP_s4752348(#   SELECT emp_no
EMP_s4752348(#   FROM employees_public
EMP_s4752348(#   ) a
EMP_s4752348(#   WHERE confidential_table_from_EMP_Confidential.emp_no = a.emp_no
EMP_s4752348(#   AND confidential_table_from_EMP_Confidential.birth_date >= '1970-01-01'
EMP_s4752348(#   AND confidential_table_from_EMP_Confidential.birth_date < '1975-01-01'
EMP_s4752348(# ) b
EMP_s4752348-# WHERE employees_public.emp_no = b.emp_no;
                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=100.84..228.13 rows=15 width=15)
   Join Filter: (confidential_table_from_emp_confidential.emp_no = employees_public.emp_no)
   ->  Nested Loop  (cost=100.42..220.78 rows=15 width=8)
         ->  Foreign Scan on confidential_table_from_emp_confidential  (cost=100.00..154.18 rows=15 width=4)
         ->  Index Only Scan using employees_public_pkey on employees_public employees_public_1  (cost=0.42..4.44 rows=1 width=4)
               Index Cond: (emp_no = confidential_table_from_emp_confidential.emp_no)
   ->  Index Scan using employees_public_pkey on employees_public  (cost=0.42..0.48 rows=1 width=19)
         Index Cond: (emp_no = employees_public_1.emp_no)
(8 rows)
```

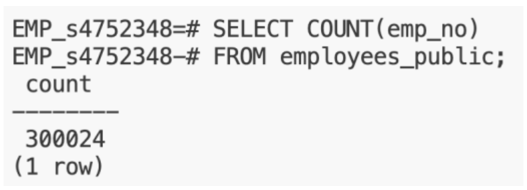


```
EMP_s4752348=# EXPLAIN SELECT employees_public.first_name, employees_public.last_name
EMP_s4752348-# FROM employees_public
EMP_s4752348-# INNER JOIN confidential_table_from_EMP_Confidential ON
EMP_s4752348-#   employees_public.emp_no = confidential_table_from_EMP_Confidential.emp_no
EMP_s4752348-# WHERE confidential_table_from_EMP_Confidential.birth_date >= '1970-01-01'
EMP_s4752348-# AND confidential_table_from_EMP_Confidential.birth_date < '1975-01-01';
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=100.42..280.78 rows=15 width=15)
   ->  Foreign Scan on confidential_table_from_emp_confidential  (cost=100.00..154.18 rows=15 width=4)
   ->  Index Scan using employees_public_pkey on employees_public  (cost=0.42..8.44 rows=1 width=19)
         Index Cond: (emp_no = confidential_table_from_emp_confidential.emp_no)
(4 rows)
```

Transmission cost:

- Semi Join:

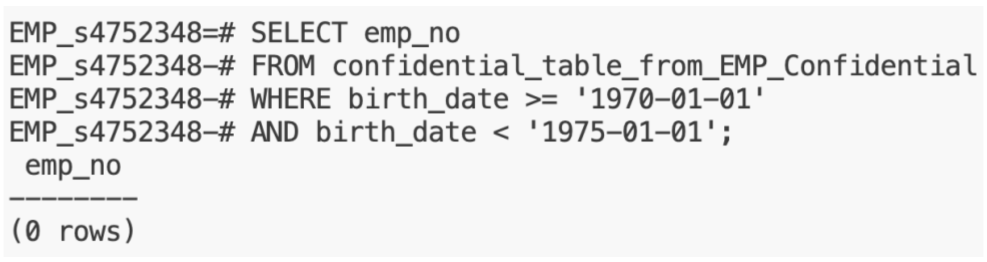
First, we could calculate the transmission cost that transfer all primary keys from local table to foreign table by following:



```
EMP_s4752348=# SELECT COUNT(emp_no)
EMP_s4752348-# FROM employees_public;
 count
--------
 300024
(1 row)
```

We could use COUNT without unique because emp_no is primary key (no duplication).

The remaining part of transmission cost is from transferring records from foreign to local site, however, in this case, there is no tuple to be return because no record is satisfied the birth date in WHERE clause. Therefore, the transmission cost is 300,024 in semi join.



```
EMP_s4752348=# SELECT emp_no
EMP_s4752348-# FROM confidential_table_from_EMP_Confidential
EMP_s4752348-# WHERE birth_date >= '1970-01-01'
EMP_s4752348-# AND birth_date < '1975-01-01';
 emp_no
--------
(0 rows)
```

- Inner join:

We know that there are also 300,024 tuples that should be transfer from foreign table to local table, however, in inner join it would transfer all of the attributes in foreign table. So, the transmission cost would be 300024 * 3 (which are emp_no, birth_date, and gender, all of them are constrained by NOT NULL) = 900,072.

To sum up, in this case, the cost of semi join is cheaper than inner join, the main reason is that the semi join only send necessary record back to local site after receiving and checking the records from local site. On the other hand, the inner join would directly send all of the records from foreign to local site. In terms of the cost calculated by psql. It also indicates that the cost of semi join is averagely cheaper than inner join.