Mahya Jamshidian 9525133

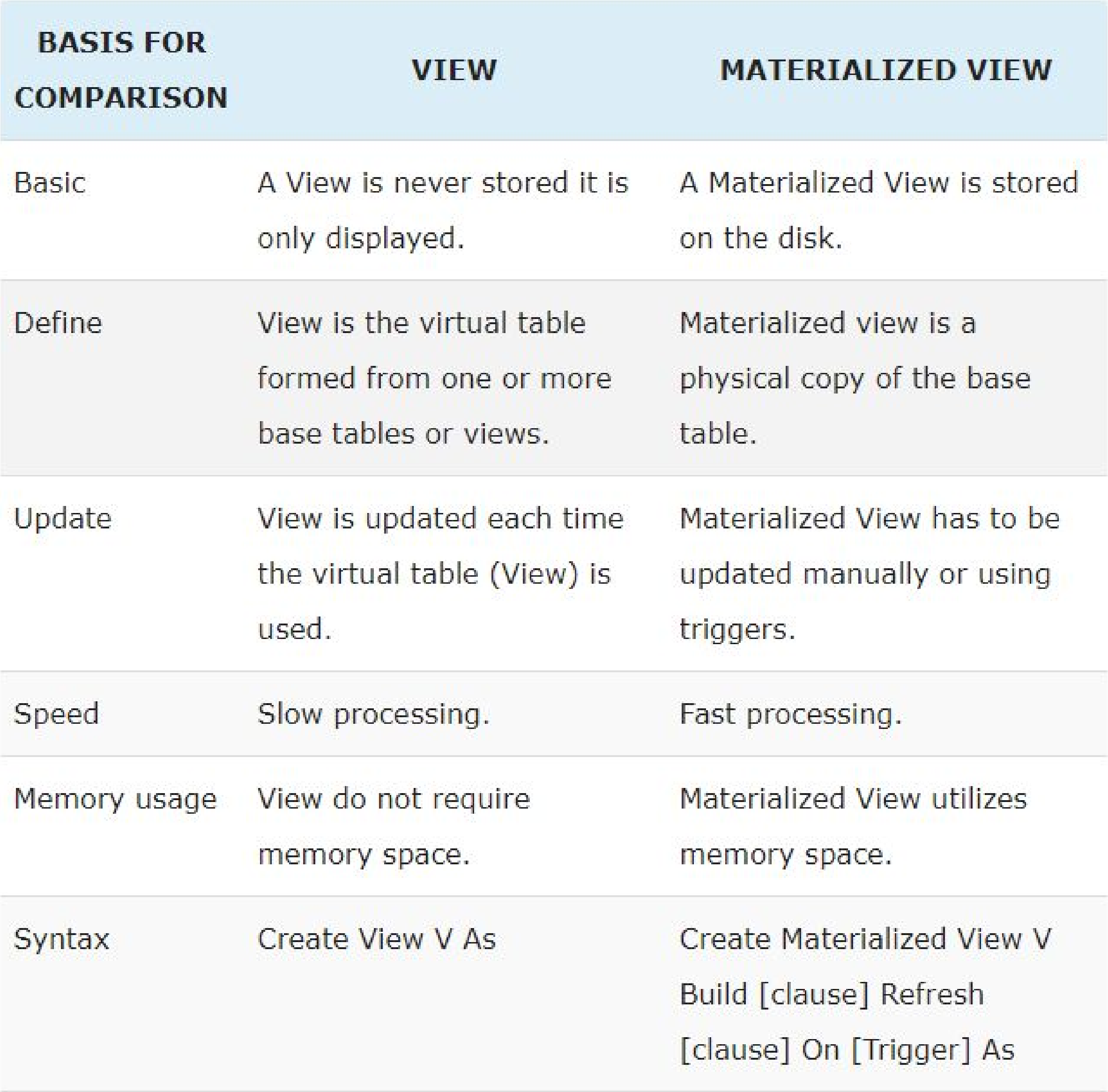
DB1 - HW3

1.
    a. **Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

       Consistency: Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

       Isolation: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

       Durability: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.
    b. Because then, this particular transaction turns out to be a bottleneck for the whole system, and hence, reduces its performance. While the overall throughput for the parallel execution of the transactions is of high importance.
    c. A serializable schedule always leaves the database in a consistent state. A serial schedule is always a serializable schedule because, in the serial schedule, a transaction only starts when the other transaction finished execution. However, a non-serial schedule needs to be checked for Serializability.
    d. Yes; Since each of the T1 or T2 to be executed first would not violate the consistency condition of the DB for exactly one of A or B is equal to one.
2.
    a. While in a materialized view we create a table and then we run queries for the new table. However, in a simple view, only some transaction is executing when the mentioned view executes (all forms of the queries are then after the temporary creation of the table).
    b. As follows



BASIS FOR COMPARISON	VIEW	MATERIALIZED VIEW
Basic	A View is never stored it is only displayed.	A Materialized View is stored on the disk.
Define	View is the virtual table formed from one or more base tables or views.	Materialized view is a physical copy of the base table.
Update	View is updated each time the virtual table (View) is used.	Materialized View has to be updated manually or using triggers.
Speed	Slow processing.	Fast processing.
Memory usage	View do not require memory space.	Materialized View utilizes memory space.
Syntax	Create View V As	Create Materialized View V Build [clause] Refresh [clause] On [Trigger] As

c. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions or window functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
- DISTINCT
- GROUP BY
- HAVING

- UNION or UNION ALL
- Subquery in the select list

  Nondependent subqueries in the select list fail for INSERT, but are okay for UPDATE, DELETE. For dependent subqueries in the select list, no data change statements are permitted.
- Certain joins (see additional join discussion later in this section)
- Reference to nonupdatable view in the FROM clause
- Subquery in the WHERE clause that refers to a table in the FROM clause
- Refers only to literal values (in this case, there is no underlying table to update)
- ALGORITHM = TEMPTABLE (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table (fails for INSERT, okay for UPDATE, DELETE)

d.

i. when the table is not there view it will not work.
ii. DML is not possible if that is more than one table.
iii. it is also a database object so it will occupy the space.
iv. When the table is dropped view becomes inactive. It depends on the table objects.
v. Querying from view takes more time than directly querying from the table

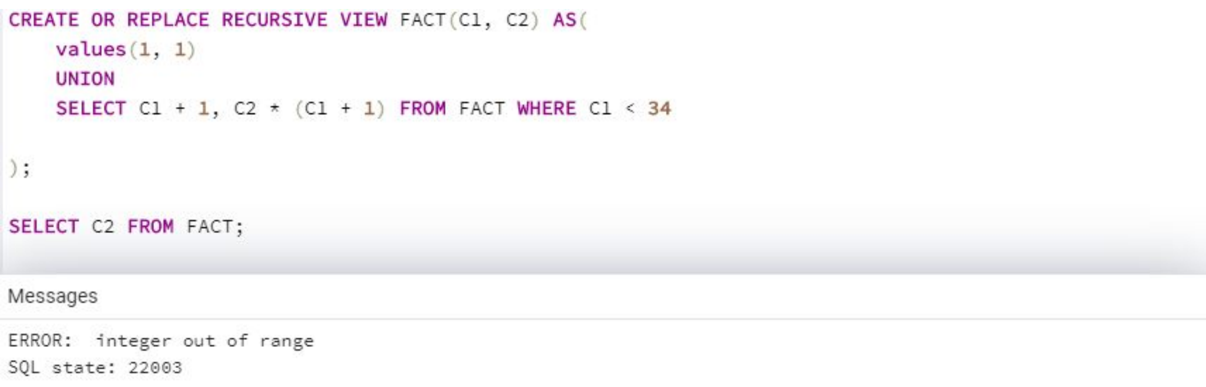
3.

Function	SP
The function always returns a value.	Stored Procedure will not return a value, but the procedure can return "0" or n values.
Functions have only input parameters for it.	Whereas, Procedures can have output or input parameters.
You can call Functions from Procedure.	But the vice-versa is not correct. As you can't call Procedures from a

	Function.
You can’t utilize Procedures in a SELECT statement.	But Function can be utilized in a SELECT statement.

4.

5. As follows, the value for 13! And higher cannot fit into SQL integer.



```
CREATE OR REPLACE RECURSIVE VIEW FACT(C1, C2) AS(
    values(1, 1)
    UNION
    SELECT C1 + 1, C2 * (C1 + 1) FROM FACT WHERE C1 < 34

);

SELECT C2 FROM FACT;
```

Messages

```
ERROR:  integer out of range
SQL state: 22003
```

6.

   a. Create `archive_tablespace` (if you want you can separate hardware on archive)

   b. Create tables. For example, we want to archive table posts.

      ```
      create table posts_all ( LIKE public.posts) ; create table
      posts_archive () inherits ( public.posts_all) ; alter table
      public.posts inherits ( public.posts_all ) ;
      ```

      After that, we will have 2 new tables: public.posts_all (with the same columns as in posts) to query all posts (archive and production) and public.posts_archive to query all archive posts. Public.posts will inherit from posts_all.

      Inserts should go in an old way (to table public.posts) unless you will write triggers on posts_all to redirect inserts to posts table. If you have partitioned it will be more complicated. With working application and before old data migration you don't have to change anything in application code to work with this approach.

   c. Create schema archive for logical separation. My suggestion will be to separate archive data by some time period (year or month) if possible (archive_2005).

d. Create archive tables in archive_year schema

```
create table archive_2005.posts ( check(record_date >=
'2005-01-01 00:00:00'::timestamp and record_date < '2006-01-01
00:00:00'::timestamp) ) inherits (posts_archive) tablespace
archive_tablesapce;
```

After that you will have new table posts in schema archive_2005 and postgresql planer will know that data there is only in designed time period. If you query by another time period postgresql will not search in this table.

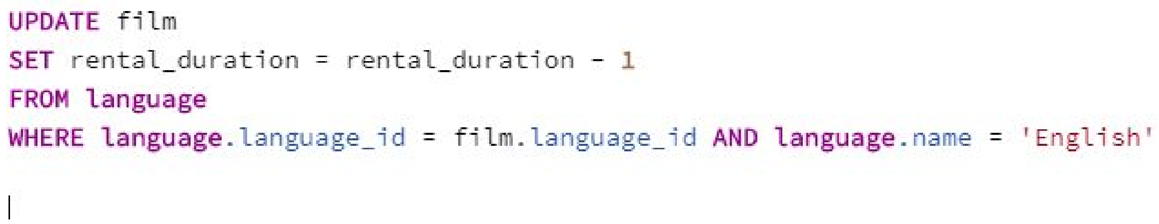
e. Create functions/procedures/triggers to move data to archive tables.

f. Do archive once for a time period (year here) and vacuum old table or do it automatically by triggers (heavier on autovacuum). There are many advantages and disadvantages to both techniques.

7.

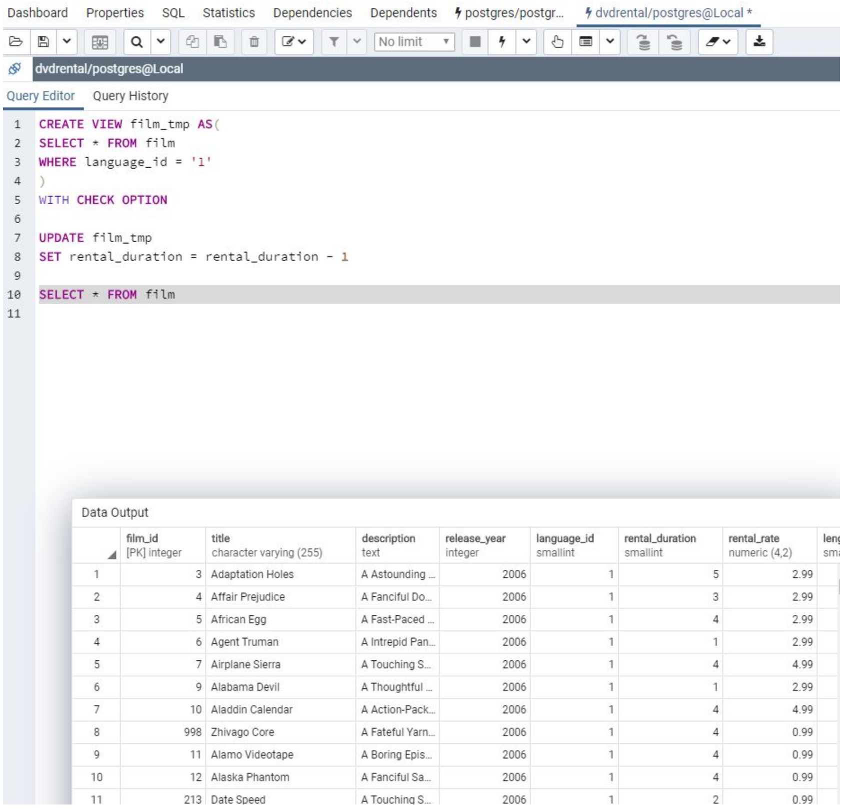
a.

b. Without view



```
UPDATE film
SET rental_duration = rental_duration - 1
FROM language
WHERE language.language_id = film.language_id AND language.name = 'English'
```

With view



```
CREATE VIEW film_tmp AS(
SELECT * FROM film
WHERE language_id = '1'
)
WITH CHECK OPTION

UPDATE film_tmp
SET rental_duration = rental_duration - 1

SELECT * FROM film

```

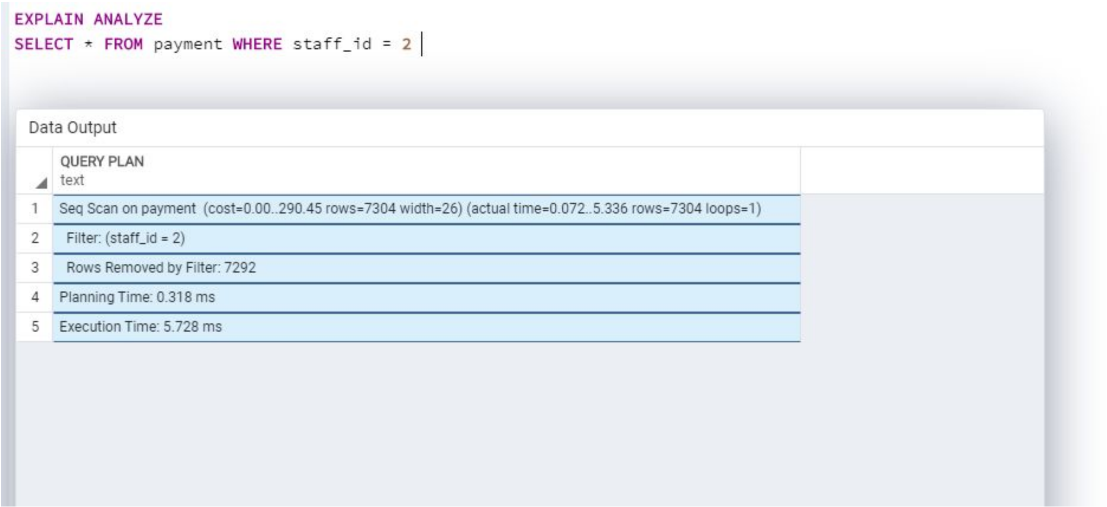
Data Output

	film_id [PK] integer	title character varying (255)	description text	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)	leng sma
1	3	Adaptation Holes	A Astounding ...	2006	1	5	2.99	
2	4	Affair Prejudice	A Fanciful Do...	2006	1	3	2.99	
3	5	African Egg	A Fast-Paced ...	2006	1	4	2.99	
4	6	Agent Truman	A Intrepid Pan...	2006	1	1	2.99	
5	7	Airplane Sierra	A Touching S...	2006	1	4	4.99	
6	9	Alabama Devil	A Thoughtful ...	2006	1	1	2.99	
7	10	Aladdin Calendar	A Action-Pack...	2006	1	4	4.99	
8	998	Zhivago Core	A Fateful Yarn...	2006	1	4	0.99	
9	11	Alamo Videotape	A Boring Epis...	2006	1	4	0.99	
10	12	Alaska Phantom	A Fanciful Sa...	2006	1	4	0.99	
11	213	Date Speed	A Touching S...	2006	1	2	0.99	

c. Either create different tables for each store physically or virtually with a view and then grant access to each vire accordingly.

8. -

a. -

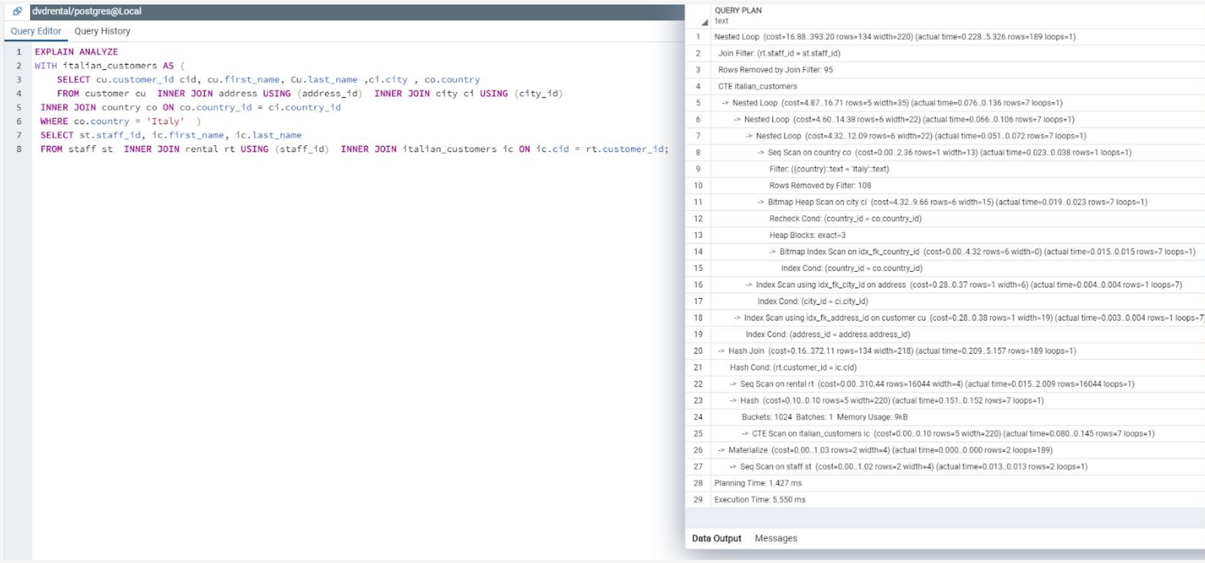


```
EXPLAIN ANALYZE
SELECT * FROM payment WHERE staff_id = 2
```

Data Output

	QUERY PLAN text
1	Seq Scan on payment (cost=0.00..290.45 rows=7304 width=26) (actual time=0.072..5.336 rows=7304 loops=1)
2	Filter: (staff_id = 2)
3	Rows Removed by Filter: 7292
4	Planning Time: 0.318 ms
5	Execution Time: 5.728 ms

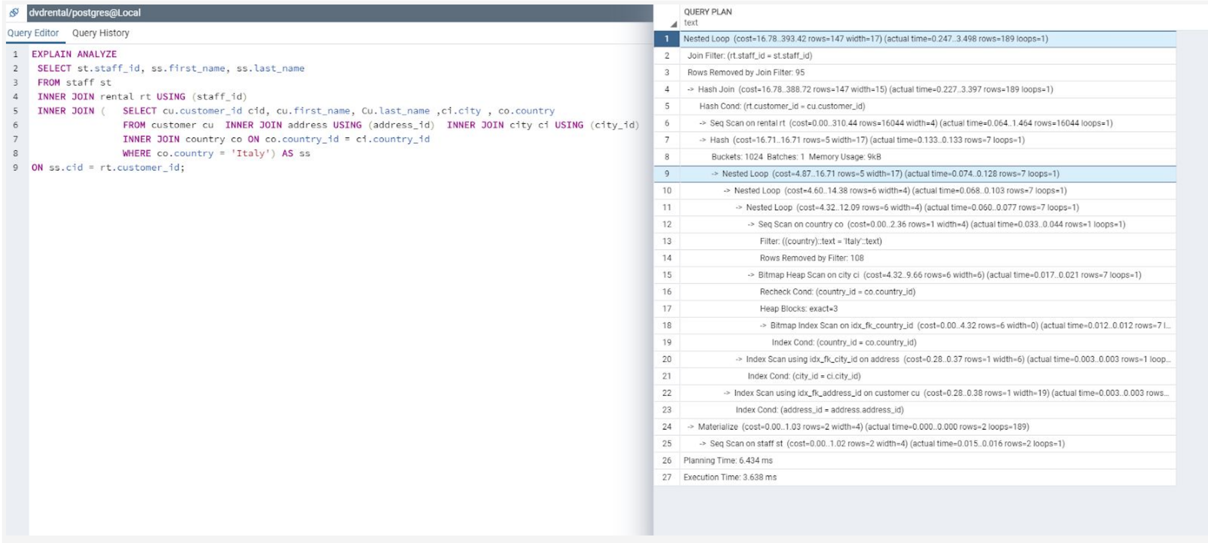
b. With statement



```sql
EXPLAIN ANALYZE
WITH italian_customers AS (
    SELECT cu.customer_id cid, cu.first_name, Cu.last_name ,ci.city , co.country
    FROM customer cu  INNER JOIN address USING (address_id)  INNER JOIN city ci USING (city_id)
 INNER JOIN country co ON co.country_id = ci.country_id
 WHERE co.country = 'Italy'  )
 SELECT st.staff_id, ic.first_name, ic.last_name
 FROM staff st  INNER JOIN rental rt USING (staff_id)  INNER JOIN italian_customers ic ON ic.cid = rt.customer_id;
```

```
   QUERY PLAN
   text
1  Nested Loop  (cost=16.88..393.20 rows=134 width=220) (actual time=0.228..5.326 rows=189 loops=1)
2    Join Filter: (rt.staff_id = st.staff_id)
3    Rows Removed by Join Filter: 95
4    CTE italian_customers
5      ->  Nested Loop  (cost=4.87..16.71 rows=5 width=35) (actual time=0.076..0.136 rows=7 loops=1)
6            ->  Nested Loop  (cost=4.60..14.38 rows=6 width=22) (actual time=0.066..0.106 rows=7 loops=1)
7                  ->  Nested Loop  (cost=4.32..12.09 rows=6 width=22) (actual time=0.051..0.072 rows=7 loops=1)
8                        ->  Seq Scan on country co  (cost=0.00..2.36 rows=1 width=13) (actual time=0.023..0.038 rows=1 loops=1)
9                              Filter: ((country)::text = 'Italy'::text)
10                             Rows Removed by Filter: 108
11                       ->  Bitmap Heap Scan on city ci  (cost=4.32..9.66 rows=6 width=15) (actual time=0.019..0.023 rows=7 loops=1)
12                             Recheck Cond: (country_id = co.country_id)
13                             Heap Blocks: exact=3
14                             ->  Bitmap Index Scan on idx_fk_country_id  (cost=0.00..4.32 rows=6 width=0) (actual time=0.015..0.015 rows=7 loops=1)
15                                   Index Cond: (country_id = co.country_id)
16                 ->  Index Scan using idx_fk_city_id on address  (cost=0.28..0.37 rows=1 width=6) (actual time=0.004..0.004 rows=1 loops=7)
17                       Index Cond: (city_id = ci.city_id)
18           ->  Index Scan using idx_fk_address_id on customer cu  (cost=0.28..0.38 rows=1 width=19) (actual time=0.003..0.004 rows=1 loops=7)
19                 Index Cond: (address_id = address.address_id)
20   ->  Hash Join  (cost=0.16..372.11 rows=134 width=218) (actual time=0.209..5.157 rows=189 loops=1)
21         Hash Cond: (rt.customer_id = ic.cid)
22         ->  Seq Scan on rental rt  (cost=0.00..310.44 rows=16044 width=4) (actual time=0.015..2.009 rows=16044 loops=1)
23         ->  Hash  (cost=0.10..0.10 rows=5 width=220) (actual time=0.151..0.152 rows=7 loops=1)
24               Buckets: 1024  Batches: 1  Memory Usage: 9kB
25               ->  CTE Scan on italian_customers ic  (cost=0.00..0.10 rows=5 width=220) (actual time=0.080..0.145 rows=7 loops=1)
26   ->  Materialize  (cost=0.00..1.03 rows=2 width=4) (actual time=0.000..0.000 rows=2 loops=189)
27         ->  Seq Scan on staff st  (cost=0.00..1.02 rows=2 width=4) (actual time=0.013..0.013 rows=2 loops=1)
28 Planning Time: 1.427 ms
29 Execution Time: 5.550 ms
```

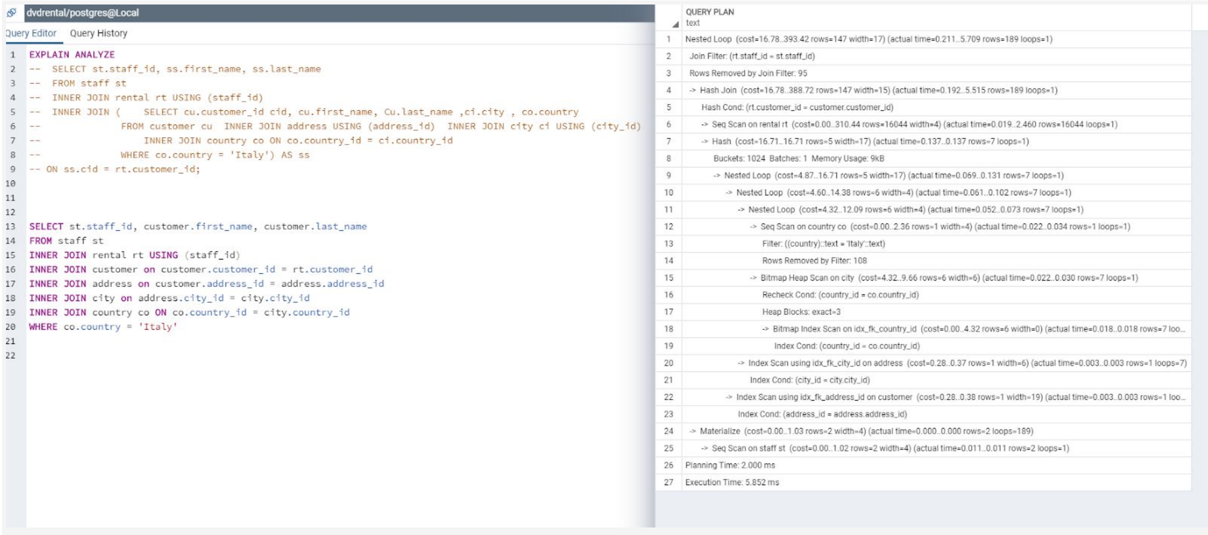
NESTED QUERY



```sql
EXPLAIN ANALYZE
 SELECT st.staff_id, ss.first_name, ss.last_name
 FROM staff st
 INNER JOIN rental rt USING (staff_id)
 INNER JOIN (   SELECT cu.customer_id cid, cu.first_name, Cu.last_name ,ci.city , co.country
                FROM customer cu  INNER JOIN address USING (address_id)  INNER JOIN city ci USING (city_id)
                INNER JOIN country co ON co.country_id = ci.country_id
                WHERE co.country = 'Italy') AS ss
ON ss.cid = rt.customer_id;
```

```
   QUERY PLAN
   text
1  Nested Loop  (cost=16.78..393.42 rows=147 width=17) (actual time=0.247..3.498 rows=189 loops=1)
2    Join Filter: (rt.staff_id = st.staff_id)
3    Rows Removed by Join Filter: 95
4    ->  Hash Join  (cost=16.78..388.72 rows=147 width=15) (actual time=0.227..3.397 rows=189 loops=1)
5          Hash Cond: (rt.customer_id = cu.customer_id)
6          ->  Seq Scan on rental rt  (cost=0.00..310.44 rows=16044 width=4) (actual time=0.064..1.464 rows=16044 loops=1)
7          ->  Hash  (cost=16.71..16.71 rows=5 width=17) (actual time=0.133..0.133 rows=7 loops=1)
8                Buckets: 1024  Batches: 1  Memory Usage: 9kB
9                ->  Nested Loop  (cost=4.87..16.71 rows=5 width=17) (actual time=0.074..0.128 rows=7 loops=1)
10                     ->  Nested Loop  (cost=4.60..14.38 rows=6 width=4) (actual time=0.068..0.103 rows=7 loops=1)
11                           ->  Nested Loop  (cost=4.32..12.09 rows=6 width=4) (actual time=0.060..0.077 rows=7 loops=1)
12                                 ->  Seq Scan on country co  (cost=0.00..2.36 rows=1 width=4) (actual time=0.033..0.044 rows=1 loops=1)
13                                       Filter: ((country)::text = 'Italy'::text)
14                                       Rows Removed by Filter: 108
15                                 ->  Bitmap Heap Scan on city ci  (cost=4.32..9.66 rows=6 width=6) (actual time=0.017..0.021 rows=7 loops=1)
16                                       Recheck Cond: (country_id = co.country_id)
17                                       Heap Blocks: exact=3
18                                       ->  Bitmap Index Scan on idx_fk_country_id  (cost=0.00..4.32 rows=6 width=0) (actual time=0.012..0.012 rows=7 l...
19                                             Index Cond: (country_id = co.country_id)
20                           ->  Index Scan using idx_fk_city_id on address  (cost=0.28..0.37 rows=1 width=6) (actual time=0.003..0.003 rows=1 loop...
21                                 Index Cond: (city_id = ci.city_id)
22                     ->  Index Scan using idx_fk_address_id on customer cu  (cost=0.28..0.38 rows=1 width=19) (actual time=0.003..0.003 rows...
23                           Index Cond: (address_id = address.address_id)
24   ->  Materialize  (cost=0.00..1.03 rows=2 width=4) (actual time=0.000..0.000 rows=2 loops=189)
25         ->  Seq Scan on staff st  (cost=0.00..1.02 rows=2 width=4) (actual time=0.015..0.016 rows=2 loops=1)
26 Planning Time: 6.434 ms
27 Execution Time: 3.638 ms
```

MORE INNER JOIN



```sql
EXPLAIN ANALYZE
--  SELECT st.staff_id, ss.first_name, ss.last_name
--  FROM staff st
--  INNER JOIN rental rt USING (staff_id)
--  INNER JOIN (    SELECT cu.customer_id cid, cu.first_name, Cu.last_name ,ci.city , co.country
--                 FROM customer cu  INNER JOIN address USING (address_id)  INNER JOIN city ci USING (city_id)
--                     INNER JOIN country co ON co.country_id = ci.country_id
--                 WHERE co.country = 'Italy') AS ss
-- ON ss.cid = rt.customer_id;



SELECT st.staff_id, customer.first_name, customer.last_name
FROM staff st
INNER JOIN rental rt USING (staff_id)
INNER JOIN customer on customer.customer_id = rt.customer_id
INNER JOIN address on customer.address_id = address.address_id
INNER JOIN city on address.city_id = city.city_id
INNER JOIN country co ON co.country_id = city.country_id
WHERE co.country = 'Italy'
```

```
   QUERY PLAN
   text
1  Nested Loop  (cost=16.78..393.42 rows=147 width=17) (actual time=0.211..5.709 rows=189 loops=1)
2    Join Filter: (rt.staff_id = st.staff_id)
3    Rows Removed by Join Filter: 95
4    ->  Hash Join  (cost=16.78..388.72 rows=147 width=15) (actual time=0.192..5.515 rows=189 loops=1)
5          Hash Cond: (rt.customer_id = customer.customer_id)
6          ->  Seq Scan on rental rt  (cost=0.00..310.44 rows=16044 width=4) (actual time=0.019..2.460 rows=16044 loops=1)
7          ->  Hash  (cost=16.71..16.71 rows=5 width=17) (actual time=0.137..0.137 rows=7 loops=1)
8                Buckets: 1024  Batches: 1  Memory Usage: 9kB
9                ->  Nested Loop  (cost=4.87..16.71 rows=5 width=17) (actual time=0.069..0.131 rows=7 loops=1)
10                     ->  Nested Loop  (cost=4.60..14.38 rows=6 width=4) (actual time=0.061..0.102 rows=7 loops=1)
11                           ->  Nested Loop  (cost=4.32..12.09 rows=6 width=4) (actual time=0.052..0.073 rows=7 loops=1)
12                                 ->  Seq Scan on country co  (cost=0.00..2.36 rows=1 width=4) (actual time=0.022..0.034 rows=1 loops=1)
13                                       Filter: ((country)::text = 'Italy'::text)
14                                       Rows Removed by Filter: 108
15                                 ->  Bitmap Heap Scan on city  (cost=4.32..9.66 rows=6 width=6) (actual time=0.022..0.030 rows=7 loops=1)
16                                       Recheck Cond: (country_id = co.country_id)
17                                       Heap Blocks: exact=3
18                                       ->  Bitmap Index Scan on idx_fk_country_id  (cost=0.00..4.32 rows=6 width=0) (actual time=0.018..0.018 rows=7 loo...
19                                             Index Cond: (country_id = co.country_id)
20                           ->  Index Scan using idx_fk_city_id on address  (cost=0.28..0.37 rows=1 width=6) (actual time=0.003..0.003 rows=1 loops=7)
21                                 Index Cond: (city_id = city.city_id)
22                     ->  Index Scan using idx_fk_address_id on customer  (cost=0.28..0.38 rows=1 width=19) (actual time=0.003..0.003 rows=1 loo...
23                           Index Cond: (address_id = address.address_id)
24   ->  Materialize  (cost=0.00..1.03 rows=2 width=4) (actual time=0.000..0.000 rows=2 loops=189)
25         ->  Seq Scan on staff st  (cost=0.00..1.02 rows=2 width=4) (actual time=0.011..0.011 rows=2 loops=1)
26 Planning Time: 2.000 ms
27 Execution Time: 5.852 ms
```