Davis Cho d2cho@ucsd.edu A11630920
Xi Yu xiy023@ucsd.edu A99076487

CSE 135 Project 2 Report

NOTE: Our SQL file is lcoated at /sqlScripts/project2.sql

ASSUMPTIONS:

1. The "Surely Beneficial Indices" decisions were made assuming tables are relatively large and it would take longer to sequentially scan tables than to index them.
2. The similarProducts page has no runtime requirements.
3. The similarProducts page returns product_id pairs.
4. Indeterministic ordering for people or states who have not bought any particular product is ordered alphabetically. The same applies to products that have not been bought.
5. The queries described in the report below are only the most resource intensive queries used in our project. A few follow-up "SELECT * FROM T…" queries were omitted as these are really quick and would not use indices as well because they simply return the "WITH T AS…" query results in a specified group by order.

-

Index Overview:

- **"Surely Beneficial Indices" decisions were made assuming tables are relatively large and it would take longer to sequentially scan than to index.**
- **In general, Query 1 and Query 3 are slower than Query 2 and Query 4 because 1 and 3 are querying far more tuples than 2 and 4 (the state filters)**
- **Overall, indices were less useful when the the percentage of rows we query from the table are very high**

Query 1:

```
WITH T AS (explain SELECT p.id AS person_id, p.person_name, pd.id AS product, pd.product_name, pic.price,
sum(pic.quantity), (pic.price*sum(pic.quantity)) AS total FROM
        shopping_cart sc
        INNER JOIN products_in_cart pic ON (pic.cart_id = sc.id)
        RIGHT OUTER JOIN product pd ON (pd.id = pic.product_id)
        RIGHT JOIN person p ON (p.id = sc.person_id)
      WHERE sc.is_purchased = 't' GROUP BY p.id, pd.id, pic.price ORDER BY p.person_name, pd.id)
```

This query is for all customers and all categories, and is repeated for a different GROUP BY and ORDER BY. Given that this selects a large percentage of rows in each table in the database, indices would prove to be less useful.

Potentially Beneficial Indices:

1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = 't', however, this is an uncertainty because realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased='t', which makes it less useful. Also for small tables, this would not be useful.

Query 2:

```
WITH T AS (explain SELECT s.id AS state_id, s.state_name, pd.id AS product, pd.product_name, pic.price,
sum(pic.quantity), (pic.price*sum(pic.quantity)) AS total FROM
        shopping_cart sc
        INNER JOIN products_in_cart pic ON (pic.cart_id = sc.id)
        RIGHT OUTER JOIN product pd ON (pd.id = pic.product_id)
        RIGHT JOIN person p ON (p.id = sc.person_id)
        INNER JOIN state s ON (s.id = p.id)
       WHERE sc.is_purchased = 't' GROUP BY s.id, pd.id, pic.price ORDER BY s.state_name, pd.id)
```

This query is for all state and all categories, and is repeated for a different GROUP BY and ORDER BY. Once again, given that this selects a large percentage of rows in each table in the database, indices would prove to be less useful.

Surely Beneficial Indices:

1. shopping_cart(person_id) v
2. products_in_cart(cart_id) v
3. person(id) default v

Potentially Beneficial Indices:

1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = ‘t’, however, this is an uncertainty because realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased=’t’, which makes it less useful. Also for small tables, this would not be useful.

Query 3:

```
WITH T AS (explain SELECT p.id AS person_id, p.person_name, c.category_name, pd.id AS product,
pd.product_name,  pic.price, sum(pic.quantity), (pic.price*sum(pic.quantity)) AS total FROM
        shopping_cart sc
        INNER JOIN products_in_cart pic ON (pic.cart_id = sc.id)
        RIGHT OUTER JOIN product pd ON (pd.id = pic.product_id)
        RIGHT JOIN person p ON (p.id = sc.person_id)
        INNER JOIN category c ON (pd.category_id = c.id)
       WHERE sc.is_purchased = 't' AND category_name = ?  GROUP BY p.id, pd.id, c.category_name, pic.price
ORDER BY p.person_name, pd.id)
```

This query is for customer and specific categories, and is repeated for a different GROUP BY and ORDER BY.

Surely Beneficial Indices:

1. shopping_cart(id) default v
2. products_in_cart(product_id) v
3. product(category_id) v
4. category(category_name) v

Potentially Beneficial Indices:

1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = ‘t’, however, this is an uncertainty because realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased=’t’, which makes it less useful. Also for small tables, this would not be useful.

Query 4:

```
WITH T AS (explain SELECT s.id AS state_id, s.state_name, c.category_name, pd.id AS product,
pd.product_name, pic.price, sum(pic.quantity), (pic.price*sum(pic.quantity)) AS total FROM
         shopping_cart sc
        INNER JOIN products_in_cart pic ON (pic.cart_id = sc.id)
        RIGHT OUTER JOIN product pd ON (pd.id = pic.product_id)
        RIGHT JOIN person p ON (p.id = sc.person_id)
        INNER JOIN state s ON (s.id = p.id)
        INNER JOIN category c ON (pd.category_id = c.id)
      WHERE sc.is_purchased = 't' AND category_name = ? GROUP BY s.id, s.state_name, c.category_name,
pd.id, pic.price ORDER BY s.state_name, pd.id)
```

This query is for state and specific categories, and is repeated for a different GROUP BY and ORDER BY.

Surely Beneficial Indices:

1. product(category_id) default v
2. person(id) default v
3. shopping_cart(id) v
4. products_in_cart(product_id) v

Potentially Beneficial Indices:

1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = ‘t’, however, this is an uncertainty because realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased=’t’, which makes it less useful. Also for small tables, this would not be useful.
2. category(category_name) default is potentially beneficial because if table is really small, it would be faster to perform a sequential scan instead.

Query 5:

```
explain SELECT p.id AS person_id, p.person_name, pd.id AS product, pd.product_name, pic.price,
sum(pic.quantity), (pic.price*sum(pic.quantity)) AS total FROM
         shopping_cart sc
        INNER JOIN products_in_cart pic ON (pic.cart_id = sc.id)
        RIGHT OUTER JOIN product pd ON (pd.id = pic.product_id)
        RIGHT JOIN person p ON (p.id = sc.person_id)
      WHERE sc.is_purchased = 't' AND p.person_name = ? AND pd.product_name = ? GROUP BY p.id, pd.id,
pic.price ORDER BY p.person_name, pd.id
```

Surely Beneficial Indices:

1. product(product_name)
2. products_in_cart(cart_id)
3. shopping_cart(person_id)
4. person(person_name) default

Potentially Beneficial Indices:

1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = ‘t’, however, this is an uncertainty because

realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased='t', which makes it less useful. Also for small tables, this would not be useful.

Query 6:
Surely Beneficial Indices:
1. product(product_name)
2. products_in_cart(cart_id)
3. shopping_cart(person_id)
4. person(id) default

Potentially Beneficial Indices:
1. shopping_cart(is_purchased) is potentially beneficial as this could help save some time searching for those that have is_purchased = 't', however, this is an uncertainty because realistically, a shopping app database containing history of shopping carts would most likely have most of the is_purchased='t', which makes it less useful. Also for small tables, this would not be useful.

-

Hot and Small Used:
100 customers, 10 categories, 100 products, 1,000 sales
Cold and Large Used:
1,000 customers, categories, and products, 500,000 sales

Part A JSP Run Times:
Hot and Small:
Run Query(customers): 480ms for first 20 rows, 400ms for subsequents
Run Query(states): 525ms for first 20 rows, 469ms for subsequents
Similar Products:
Large and Cold:
Queries Runtime(customers): 107,453ms for first 20 rows, 55,131ms for subsequents
Queries Runtime(state): 8,082ms for first 20 rows, 4,750ms for subsequents

----
-

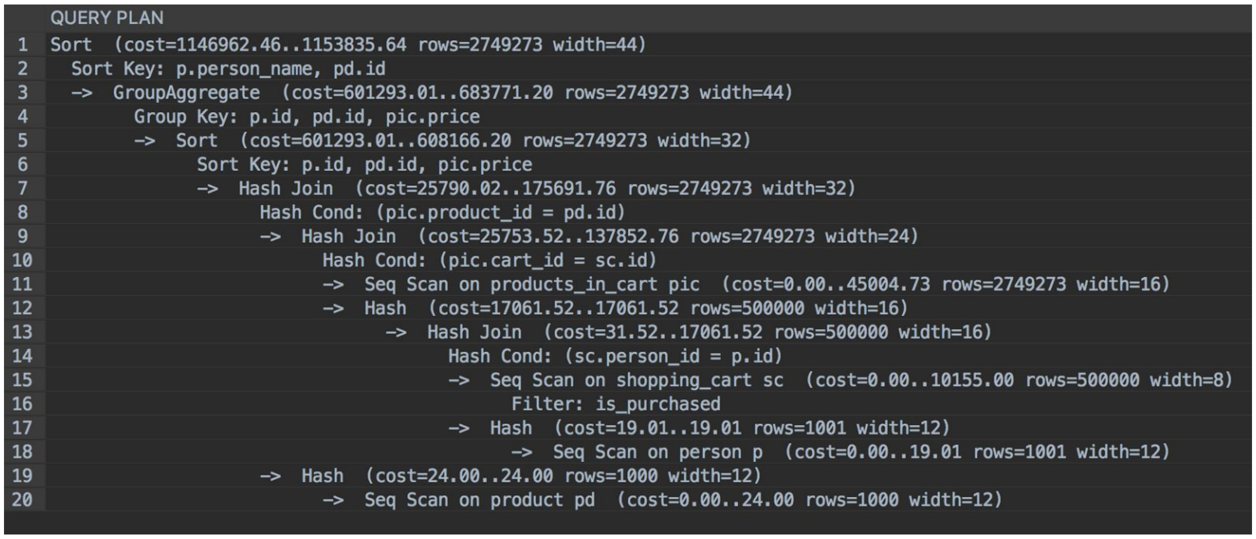
Part B Query Run Times(averaged over 3 runs):
Query 1:
Hot and Small: 71ms
Cold and Large: 19.58s
Query 2:
Hot and Small: 49ms
Cold and Large: 1.71s
Query 3:
Hot and Small: 35ms
Cold and Large: 78ms

Query 4:
Hot and Small: 29ms
Cold and Large: 53ms
Query 5:
Hot and Small: 35ms
Cold and Large: 38ms
Query 6:
Hot and Small: 32ms
Cold and Large: 31ms

-

Part C Index Choices:
As described above before Part A, any index that I predicted would not be useful when the table is small. It is also interesting to note that search conditions on low cardinality tables did not seem to be used, like the is_purchased column sometimes. Additionally, the state(id) was not used in small nor large tables because the state table itself is always a set small size. For small databases, the general trend was that specific lookups in the WHERE clause tended to be searched for sequentially rather than indexed.

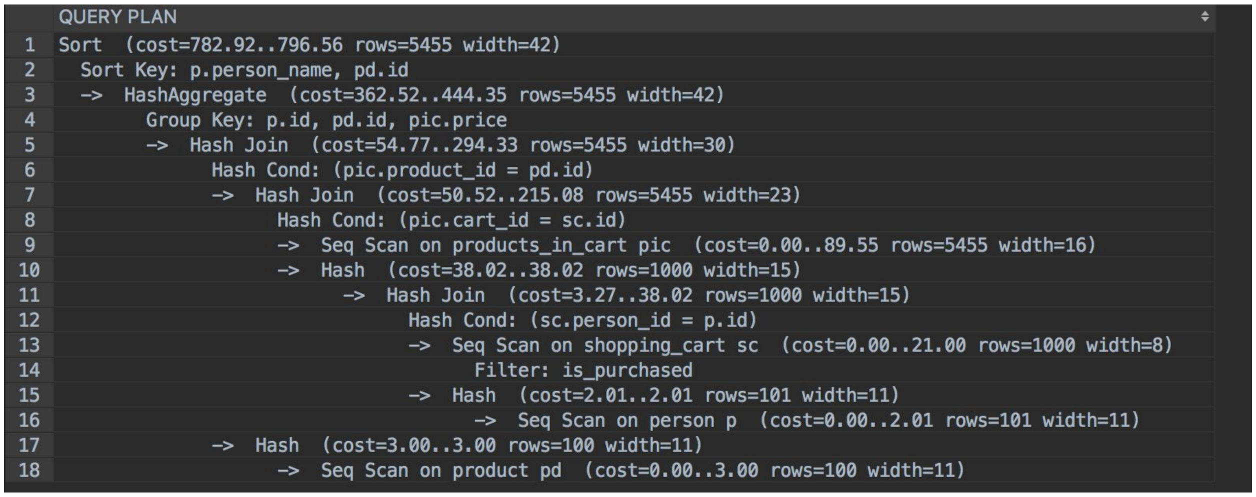
Query 1:
Cold
We select most of every table, so indices are not useful.



```
QUERY PLAN
Sort  (cost=1146962.46..1153835.64 rows=2749273 width=44)
  Sort Key: p.person_name, pd.id
  ->  GroupAggregate  (cost=601293.01..683771.20 rows=2749273 width=44)
        Group Key: p.id, pd.id, pic.price
        ->  Sort  (cost=601293.01..608166.20 rows=2749273 width=32)
              Sort Key: p.id, pd.id, pic.price
              ->  Hash Join  (cost=25790.02..175691.76 rows=2749273 width=32)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Hash Join  (cost=25753.52..137852.76 rows=2749273 width=24)
                          Hash Cond: (pic.cart_id = sc.id)
                          ->  Seq Scan on products_in_cart pic  (cost=0.00..45004.73 rows=2749273 width=16)
                          ->  Hash  (cost=17061.52..17061.52 rows=500000 width=16)
                                ->  Hash Join  (cost=31.52..17061.52 rows=500000 width=16)
                                      Hash Cond: (sc.person_id = p.id)
                                      ->  Seq Scan on shopping_cart sc  (cost=0.00..10155.00 rows=500000 width=8)
                                            Filter: is_purchased
                                      ->  Hash  (cost=19.01..19.01 rows=1001 width=12)
                                            ->  Seq Scan on person p  (cost=0.00..19.01 rows=1001 width=12)
                    ->  Hash  (cost=24.00..24.00 rows=1000 width=12)
                          ->  Seq Scan on product pd  (cost=0.00..24.00 rows=1000 width=12)
```

Hot:

The same logic applies as the cold database. Additionally, for small tables, sequential scans are quicker.

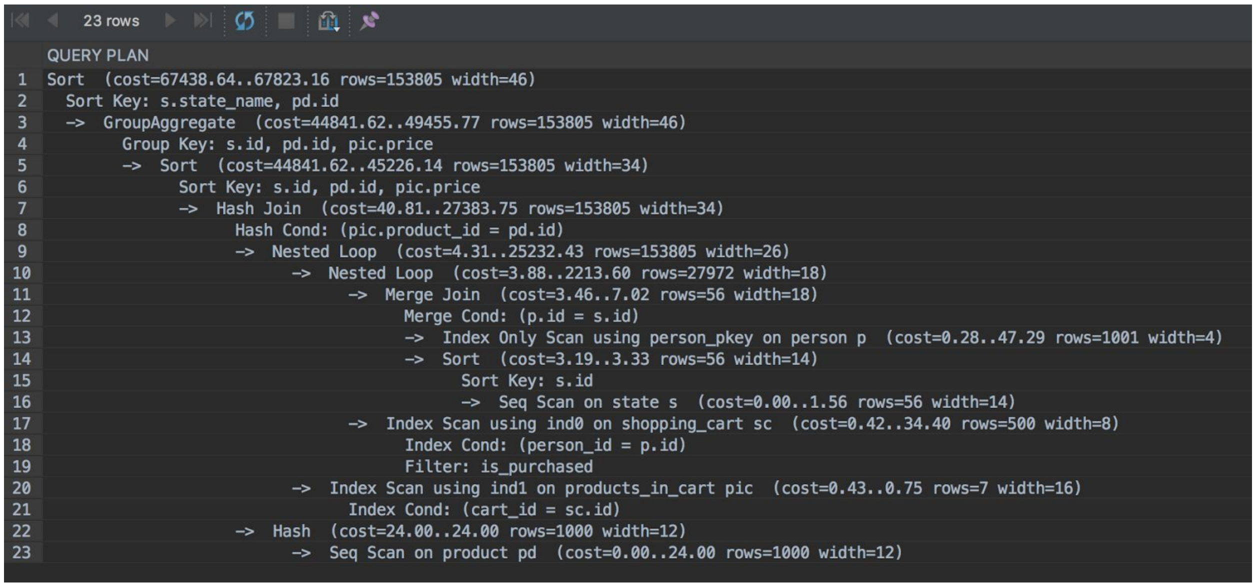


```
QUERY PLAN
Sort  (cost=782.92..796.56 rows=5455 width=42)
  Sort Key: p.person_name, pd.id
  ->  HashAggregate  (cost=362.52..444.35 rows=5455 width=42)
        Group Key: p.id, pd.id, pic.price
        ->  Hash Join  (cost=54.77..294.33 rows=5455 width=30)
              Hash Cond: (pic.product_id = pd.id)
              ->  Hash Join  (cost=50.52..215.08 rows=5455 width=23)
                    Hash Cond: (pic.cart_id = sc.id)
                    ->  Seq Scan on products_in_cart pic  (cost=0.00..89.55 rows=5455 width=16)
                    ->  Hash  (cost=38.02..38.02 rows=1000 width=15)
                          ->  Hash Join  (cost=3.27..38.02 rows=1000 width=15)
                                Hash Cond: (sc.person_id = p.id)
                                ->  Seq Scan on shopping_cart sc  (cost=0.00..21.00 rows=1000 width=8)
                                      Filter: is_purchased
                                ->  Hash  (cost=2.01..2.01 rows=101 width=11)
                                      ->  Seq Scan on person p  (cost=0.00..2.01 rows=101 width=11)
              ->  Hash  (cost=3.00..3.00 rows=100 width=11)
                    ->  Seq Scan on product pd  (cost=0.00..3.00 rows=100 width=11)
```

Query 2:

Cold:

1. person(id) default v
2. products_in_cart(cart_id) v
3. shopping_cart(person_id) v

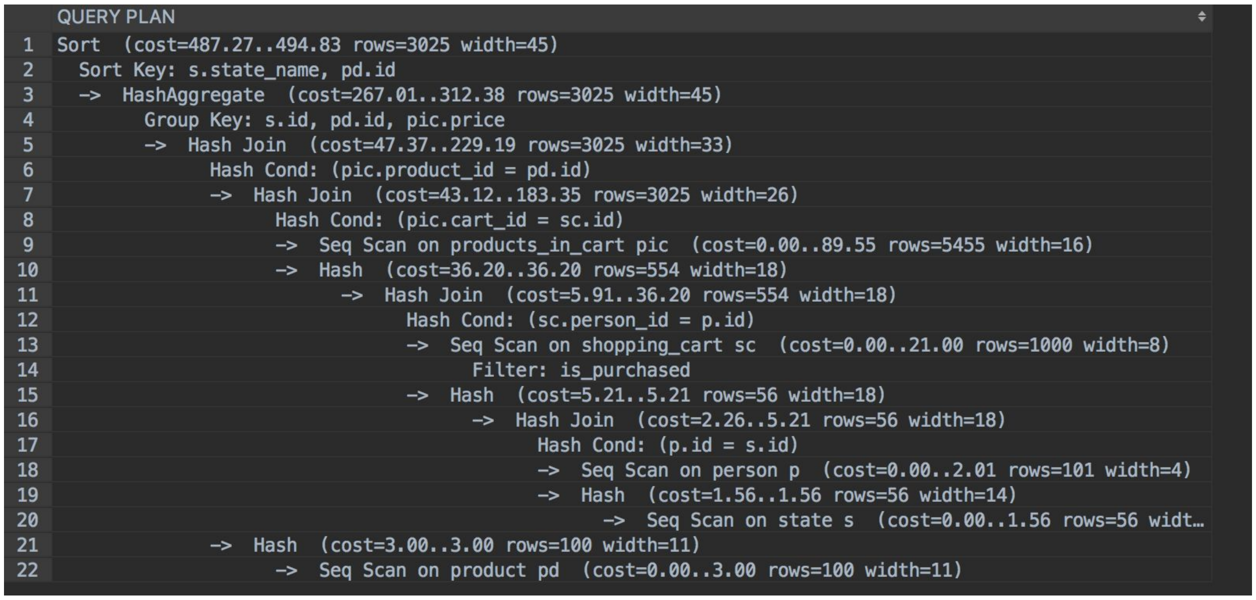


23 rows

```
QUERY PLAN
Sort  (cost=67438.64..67823.16 rows=153805 width=46)
  Sort Key: s.state_name, pd.id
  ->  GroupAggregate  (cost=44841.62..49455.77 rows=153805 width=46)
        Group Key: s.id, pd.id, pic.price
        ->  Sort  (cost=44841.62..45226.14 rows=153805 width=34)
              Sort Key: s.id, pd.id, pic.price
              ->  Hash Join  (cost=40.81..27383.75 rows=153805 width=34)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Nested Loop  (cost=4.31..25232.43 rows=153805 width=26)
                          ->  Nested Loop  (cost=3.88..2213.60 rows=27972 width=18)
                                ->  Merge Join  (cost=3.46..7.02 rows=56 width=18)
                                      Merge Cond: (p.id = s.id)
                                      ->  Index Only Scan using person_pkey on person p  (cost=0.28..47.29 rows=1001 width=4)
                                      ->  Sort  (cost=3.19..3.33 rows=56 width=14)
                                            Sort Key: s.id
                                            ->  Seq Scan on state s  (cost=0.00..1.56 rows=56 width=14)
                                ->  Index Scan using ind0 on shopping_cart sc  (cost=0.42..34.40 rows=500 width=8)
                                      Index Cond: (person_id = p.id)
                                      Filter: is_purchased
                          ->  Index Scan using ind1 on products_in_cart pic  (cost=0.43..0.75 rows=7 width=16)
                                Index Cond: (cart_id = sc.id)
                    ->  Hash  (cost=24.00..24.00 rows=1000 width=12)
                          ->  Seq Scan on product pd  (cost=0.00..24.00 rows=1000 width=12)
```

Hot:

For small tables, sequential scans run faster than indexing, so none.

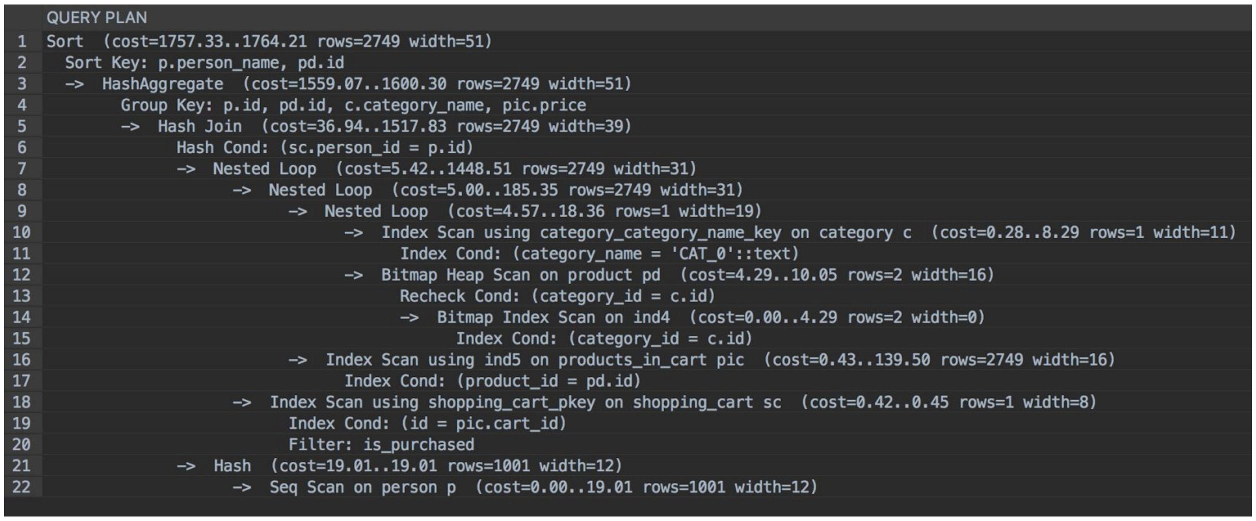


```
QUERY PLAN
Sort  (cost=487.27..494.83 rows=3025 width=45)
  Sort Key: s.state_name, pd.id
  ->  HashAggregate  (cost=267.01..312.38 rows=3025 width=45)
        Group Key: s.id, pd.id, pic.price
        ->  Hash Join  (cost=47.37..229.19 rows=3025 width=33)
              Hash Cond: (pic.product_id = pd.id)
              ->  Hash Join  (cost=43.12..183.35 rows=3025 width=26)
                    Hash Cond: (pic.cart_id = sc.id)
                    ->  Seq Scan on products_in_cart pic  (cost=0.00..89.55 rows=5455 width=16)
                    ->  Hash  (cost=36.20..36.20 rows=554 width=18)
                          ->  Hash Join  (cost=5.91..36.20 rows=554 width=18)
                                Hash Cond: (sc.person_id = p.id)
                                ->  Seq Scan on shopping_cart sc  (cost=0.00..21.00 rows=1000 width=8)
                                      Filter: is_purchased
                                ->  Hash  (cost=5.21..5.21 rows=56 width=18)
                                      ->  Hash Join  (cost=2.26..5.21 rows=56 width=18)
                                            Hash Cond: (p.id = s.id)
                                            ->  Seq Scan on person p  (cost=0.00..2.01 rows=101 width=4)
                                            ->  Hash  (cost=1.56..1.56 rows=56 width=14)
                                                  ->  Seq Scan on state s  (cost=0.00..1.56 rows=56 widt…
              ->  Hash  (cost=3.00..3.00 rows=100 width=11)
                    ->  Seq Scan on product pd  (cost=0.00..3.00 rows=100 width=11)
```

Query 3:

Cold

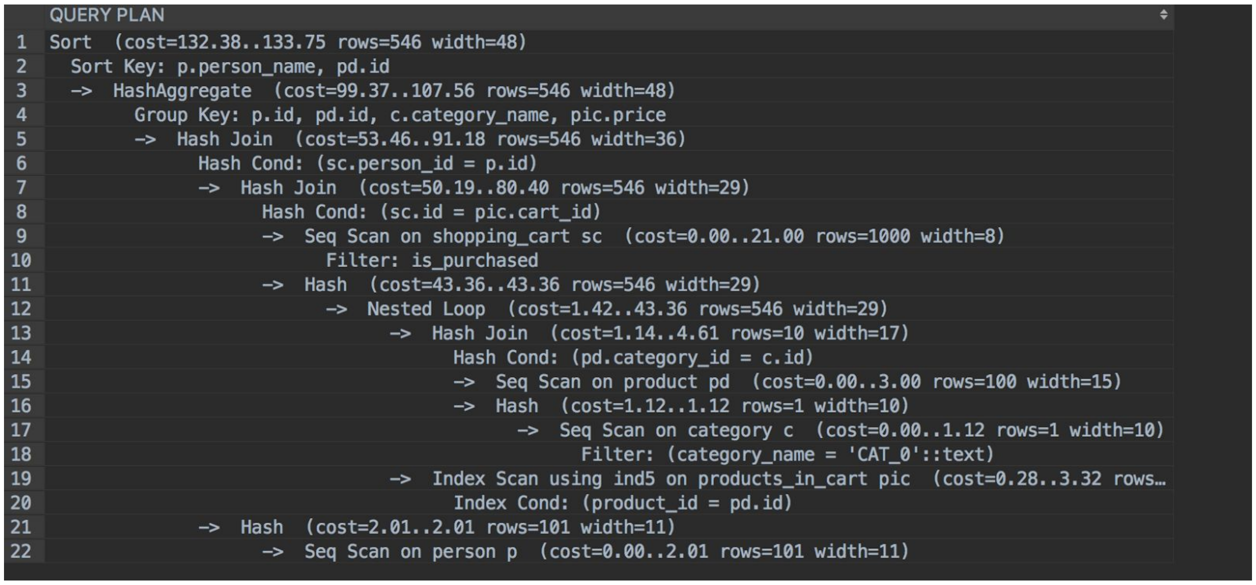
1. category(category_name) v
2. product(category_id) v
3. products_in_cart(product_id) v
4. shopping_cart(id) v



```
QUERY PLAN
Sort  (cost=1757.33..1764.21 rows=2749 width=51)
  Sort Key: p.person_name, pd.id
  ->  HashAggregate  (cost=1559.07..1600.30 rows=2749 width=51)
        Group Key: p.id, pd.id, c.category_name, pic.price
        ->  Hash Join  (cost=36.94..1517.83 rows=2749 width=39)
              Hash Cond: (sc.person_id = p.id)
              ->  Nested Loop  (cost=5.42..1448.51 rows=2749 width=31)
                    ->  Nested Loop  (cost=5.00..185.35 rows=2749 width=31)
                          ->  Nested Loop  (cost=4.57..18.36 rows=1 width=19)
                                ->  Index Scan using category_category_name_key on category c  (cost=0.28..8.29 rows=1 width=11)
                                      Index Cond: (category_name = 'CAT_0'::text)
                                ->  Bitmap Heap Scan on product pd  (cost=4.29..10.05 rows=2 width=16)
                                      Recheck Cond: (category_id = c.id)
                                      ->  Bitmap Index Scan on ind4  (cost=0.00..4.29 rows=2 width=0)
                                            Index Cond: (category_id = c.id)
                          ->  Index Scan using ind5 on products_in_cart pic  (cost=0.43..139.50 rows=2749 width=16)
                                Index Cond: (product_id = pd.id)
                    ->  Index Scan using shopping_cart_pkey on shopping_cart sc  (cost=0.42..0.45 rows=1 width=8)
                          Index Cond: (id = pic.cart_id)
                          Filter: is_purchased
              ->  Hash  (cost=19.01..19.01 rows=1001 width=12)
                    ->  Seq Scan on person p  (cost=0.00..19.01 rows=1001 width=12)
```

Hot:

1. products_in_cart(product_id) v

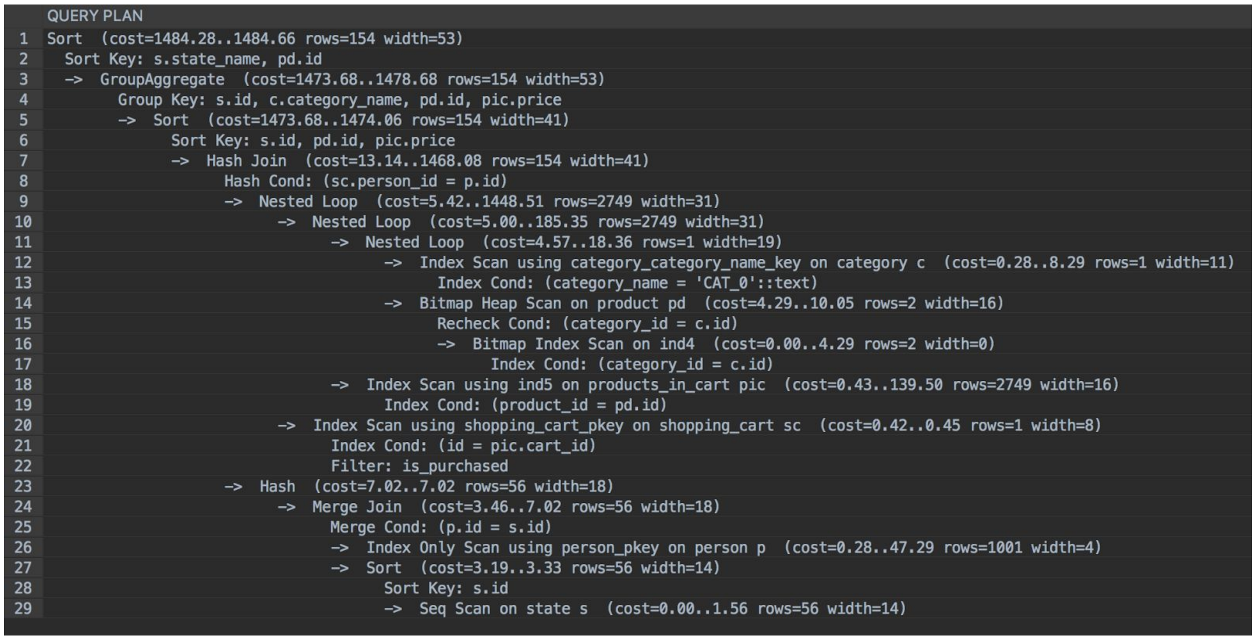


```
QUERY PLAN
Sort  (cost=132.38..133.75 rows=546 width=48)
  Sort Key: p.person_name, pd.id
  ->  HashAggregate  (cost=99.37..107.56 rows=546 width=48)
        Group Key: p.id, pd.id, c.category_name, pic.price
        ->  Hash Join  (cost=53.46..91.18 rows=546 width=36)
              Hash Cond: (sc.person_id = p.id)
              ->  Hash Join  (cost=50.19..80.40 rows=546 width=29)
                    Hash Cond: (sc.id = pic.cart_id)
                    ->  Seq Scan on shopping_cart sc  (cost=0.00..21.00 rows=1000 width=8)
                          Filter: is_purchased
                    ->  Hash  (cost=43.36..43.36 rows=546 width=29)
                          ->  Nested Loop  (cost=1.42..43.36 rows=546 width=29)
                                ->  Hash Join  (cost=1.14..4.61 rows=10 width=17)
                                      Hash Cond: (pd.category_id = c.id)
                                      ->  Seq Scan on product pd  (cost=0.00..3.00 rows=100 width=15)
                                      ->  Hash  (cost=1.12..1.12 rows=1 width=10)
                                            ->  Seq Scan on category c  (cost=0.00..1.12 rows=1 width=10)
                                                  Filter: (category_name = 'CAT_0'::text)
                                ->  Index Scan using ind5 on products_in_cart pic  (cost=0.28..3.32 rows…
                                      Index Cond: (product_id = pd.id)
              ->  Hash  (cost=2.01..2.01 rows=101 width=11)
                    ->  Seq Scan on person p  (cost=0.00..2.01 rows=101 width=11)
```

Query 4:

Cold

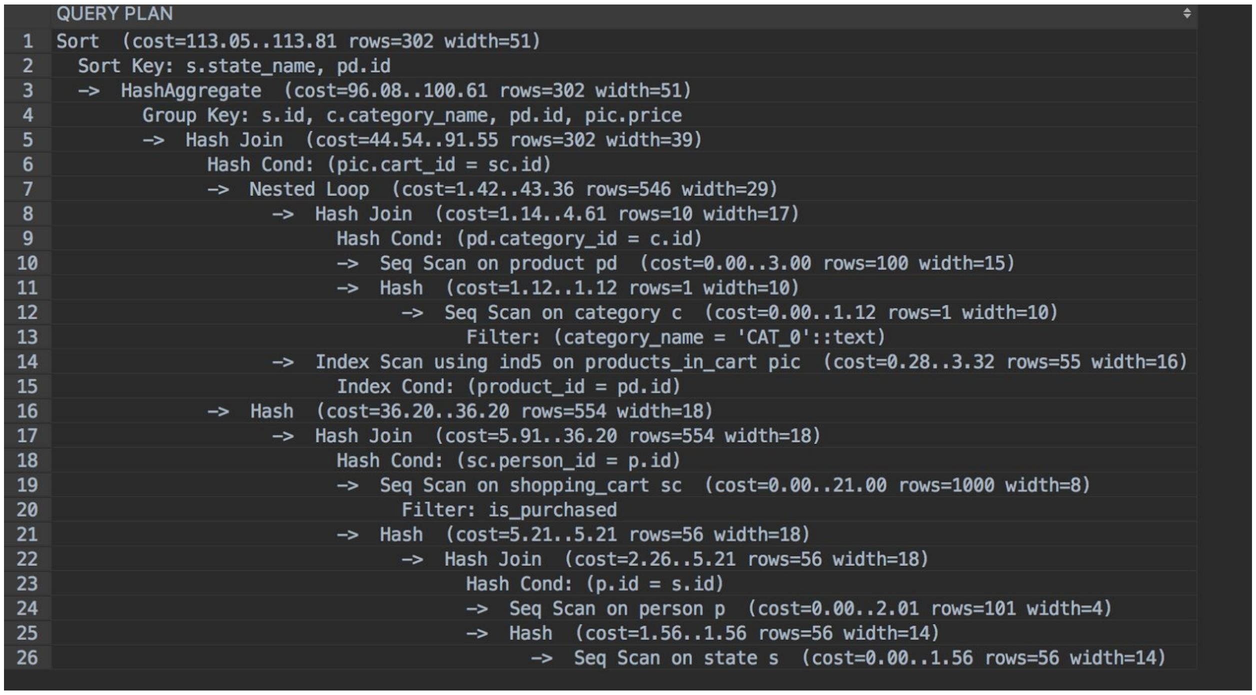
1. product(caegory_id) v
2. person(id) default v
3. shopping_cart(id) v
4. products_in_cart(product_id) v
5. category(category_name) default v



```
QUERY PLAN
Sort  (cost=1484.28..1484.66 rows=154 width=53)
  Sort Key: s.state_name, pd.id
  ->  GroupAggregate  (cost=1473.68..1478.68 rows=154 width=53)
        Group Key: s.id, c.category_name, pd.id, pic.price
        ->  Sort  (cost=1473.68..1474.06 rows=154 width=41)
              Sort Key: s.id, pd.id, pic.price
              ->  Hash Join  (cost=13.14..1468.08 rows=154 width=41)
                    Hash Cond: (sc.person_id = p.id)
                    ->  Nested Loop  (cost=5.42..1448.51 rows=2749 width=31)
                          ->  Nested Loop  (cost=5.00..185.35 rows=2749 width=31)
                                ->  Nested Loop  (cost=4.57..18.36 rows=1 width=19)
                                      ->  Index Scan using category_category_name_key on category c  (cost=0.28..8.29 rows=1 width=11)
                                            Index Cond: (category_name = 'CAT_0'::text)
                                      ->  Bitmap Heap Scan on product pd  (cost=4.29..10.05 rows=2 width=16)
                                            Recheck Cond: (category_id = c.id)
                                            ->  Bitmap Index Scan on ind4  (cost=0.00..4.29 rows=2 width=0)
                                                  Index Cond: (category_id = c.id)
                                ->  Index Scan using ind5 on products_in_cart pic  (cost=0.43..139.50 rows=2749 width=16)
                                      Index Cond: (product_id = pd.id)
                          ->  Index Scan using shopping_cart_pkey on shopping_cart sc  (cost=0.42..0.45 rows=1 width=8)
                                Index Cond: (id = pic.cart_id)
                                Filter: is_purchased
                    ->  Hash  (cost=7.02..7.02 rows=56 width=18)
                          ->  Merge Join  (cost=3.46..7.02 rows=56 width=18)
                                Merge Cond: (p.id = s.id)
                                ->  Index Only Scan using person_pkey on person p  (cost=0.28..47.29 rows=1001 width=4)
                                ->  Sort  (cost=3.19..3.33 rows=56 width=14)
                                      Sort Key: s.id
                                      ->  Seq Scan on state s  (cost=0.00..1.56 rows=56 width=14)
```

Hot:

1. products_in_cart(product_id) v



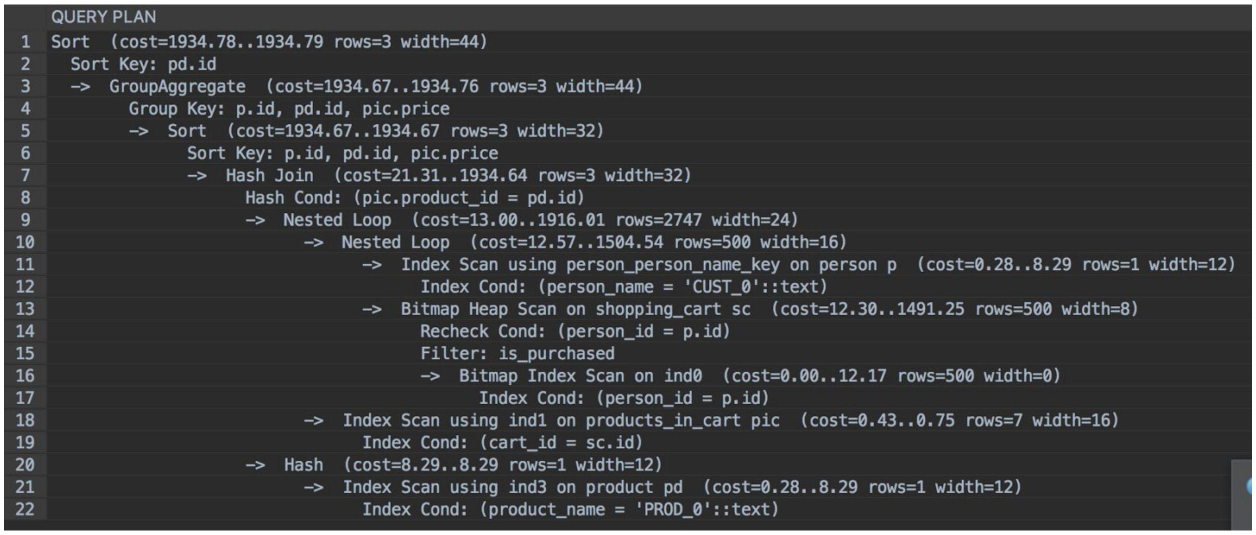
```
QUERY PLAN
Sort  (cost=113.05..113.81 rows=302 width=51)
  Sort Key: s.state_name, pd.id
  ->  HashAggregate  (cost=96.08..100.61 rows=302 width=51)
        Group Key: s.id, c.category_name, pd.id, pic.price
        ->  Hash Join  (cost=44.54..91.55 rows=302 width=39)
              Hash Cond: (pic.cart_id = sc.id)
              ->  Nested Loop  (cost=1.42..43.36 rows=546 width=29)
                    ->  Hash Join  (cost=1.14..4.61 rows=10 width=17)
                          Hash Cond: (pd.category_id = c.id)
                          ->  Seq Scan on product pd  (cost=0.00..3.00 rows=100 width=15)
                          ->  Hash  (cost=1.12..1.12 rows=1 width=10)
                                ->  Seq Scan on category c  (cost=0.00..1.12 rows=1 width=10)
                                      Filter: (category_name = 'CAT_0'::text)
                    ->  Index Scan using ind5 on products_in_cart pic  (cost=0.28..3.32 rows=55 width=16)
                          Index Cond: (product_id = pd.id)
              ->  Hash  (cost=36.20..36.20 rows=554 width=18)
                    ->  Hash Join  (cost=5.91..36.20 rows=554 width=18)
                          Hash Cond: (sc.person_id = p.id)
                          ->  Seq Scan on shopping_cart sc  (cost=0.00..21.00 rows=1000 width=8)
                                Filter: is_purchased
                          ->  Hash  (cost=5.21..5.21 rows=56 width=18)
                                ->  Hash Join  (cost=2.26..5.21 rows=56 width=18)
                                      Hash Cond: (p.id = s.id)
                                      ->  Seq Scan on person p  (cost=0.00..2.01 rows=101 width=4)
                                      ->  Hash  (cost=1.56..1.56 rows=56 width=14)
                                            ->  Seq Scan on state s  (cost=0.00..1.56 rows=56 width=14)
```

Query 5:

Cold:

1. product(product_name)
2. products_in_cart(cart_id)

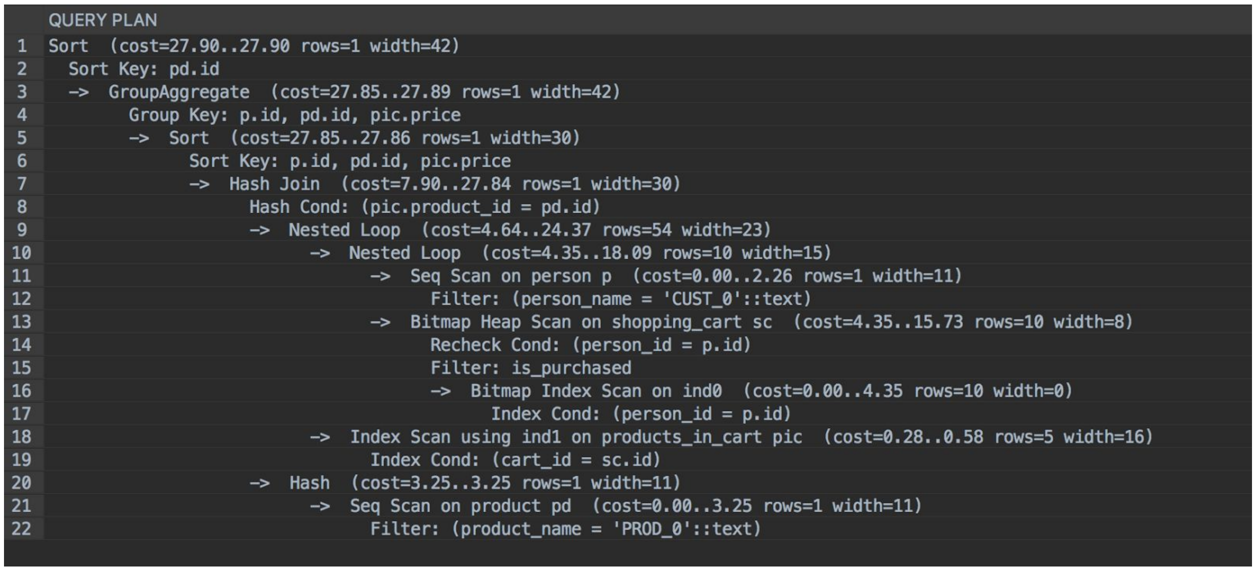
3. shopping_cart(person_id)
4. person(person_name) default



```
QUERY PLAN
Sort  (cost=1934.78..1934.79 rows=3 width=44)
  Sort Key: pd.id
  ->  GroupAggregate  (cost=1934.67..1934.76 rows=3 width=44)
        Group Key: p.id, pd.id, pic.price
        ->  Sort  (cost=1934.67..1934.67 rows=3 width=32)
              Sort Key: p.id, pd.id, pic.price
              ->  Hash Join  (cost=21.31..1934.64 rows=3 width=32)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Nested Loop  (cost=13.00..1916.01 rows=2747 width=24)
                          ->  Nested Loop  (cost=12.57..1504.54 rows=500 width=16)
                                ->  Index Scan using person_person_name_key on person p  (cost=0.28..8.29 rows=1 width=12)
                                      Index Cond: (person_name = 'CUST_0'::text)
                                ->  Bitmap Heap Scan on shopping_cart sc  (cost=12.30..1491.25 rows=500 width=8)
                                      Recheck Cond: (person_id = p.id)
                                      Filter: is_purchased
                                      ->  Bitmap Index Scan on ind0  (cost=0.00..12.17 rows=500 width=0)
                                            Index Cond: (person_id = p.id)
                          ->  Index Scan using ind1 on products_in_cart pic  (cost=0.43..0.75 rows=7 width=16)
                                Index Cond: (cart_id = sc.id)
                    ->  Hash  (cost=8.29..8.29 rows=1 width=12)
                          ->  Index Scan using ind3 on product pd  (cost=0.28..8.29 rows=1 width=12)
                                Index Cond: (product_name = 'PROD_0'::text)
```

Hot:

1. products_in_cart(cart_id) v
2. shopping_cart(person_id) v

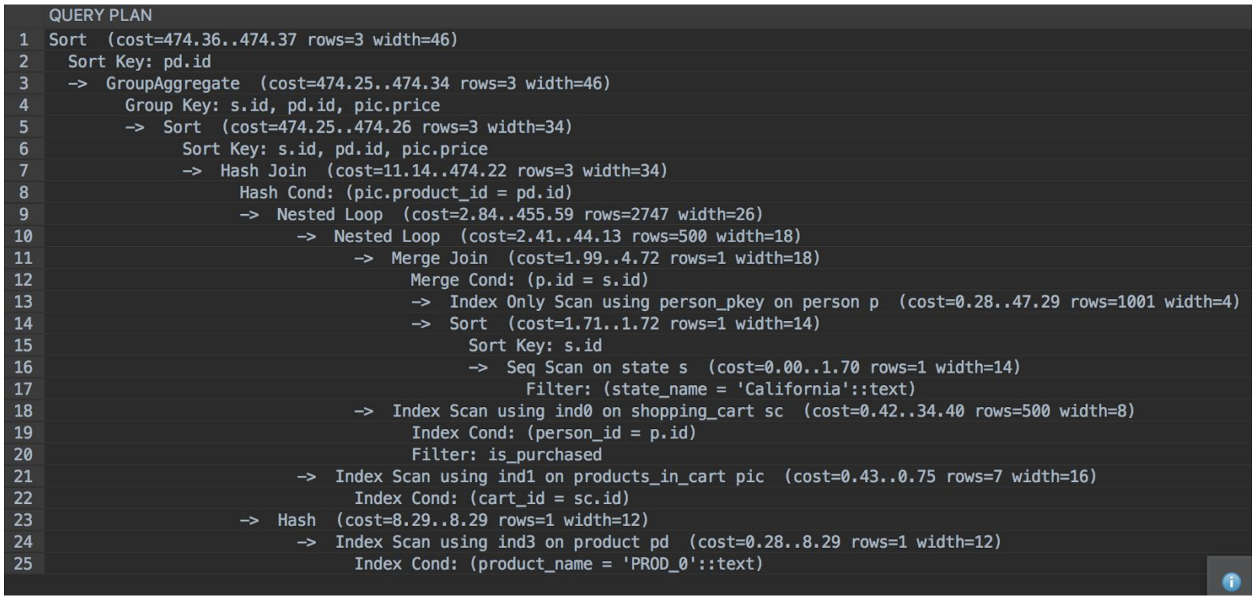


```
QUERY PLAN
Sort  (cost=27.90..27.90 rows=1 width=42)
  Sort Key: pd.id
  ->  GroupAggregate  (cost=27.85..27.89 rows=1 width=42)
        Group Key: p.id, pd.id, pic.price
        ->  Sort  (cost=27.85..27.86 rows=1 width=30)
              Sort Key: p.id, pd.id, pic.price
              ->  Hash Join  (cost=7.90..27.84 rows=1 width=30)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Nested Loop  (cost=4.64..24.37 rows=54 width=23)
                          ->  Nested Loop  (cost=4.35..18.09 rows=10 width=15)
                                ->  Seq Scan on person p  (cost=0.00..2.26 rows=1 width=11)
                                      Filter: (person_name = 'CUST_0'::text)
                                ->  Bitmap Heap Scan on shopping_cart sc  (cost=4.35..15.73 rows=10 width=8)
                                      Recheck Cond: (person_id = p.id)
                                      Filter: is_purchased
                                      ->  Bitmap Index Scan on ind0  (cost=0.00..4.35 rows=10 width=0)
                                            Index Cond: (person_id = p.id)
                          ->  Index Scan using ind1 on products_in_cart pic  (cost=0.28..0.58 rows=5 width=16)
                                Index Cond: (cart_id = sc.id)
                    ->  Hash  (cost=3.25..3.25 rows=1 width=11)
                          ->  Seq Scan on product pd  (cost=0.00..3.25 rows=1 width=11)
                                Filter: (product_name = 'PROD_0'::text)
```

Query 6:

Cold:

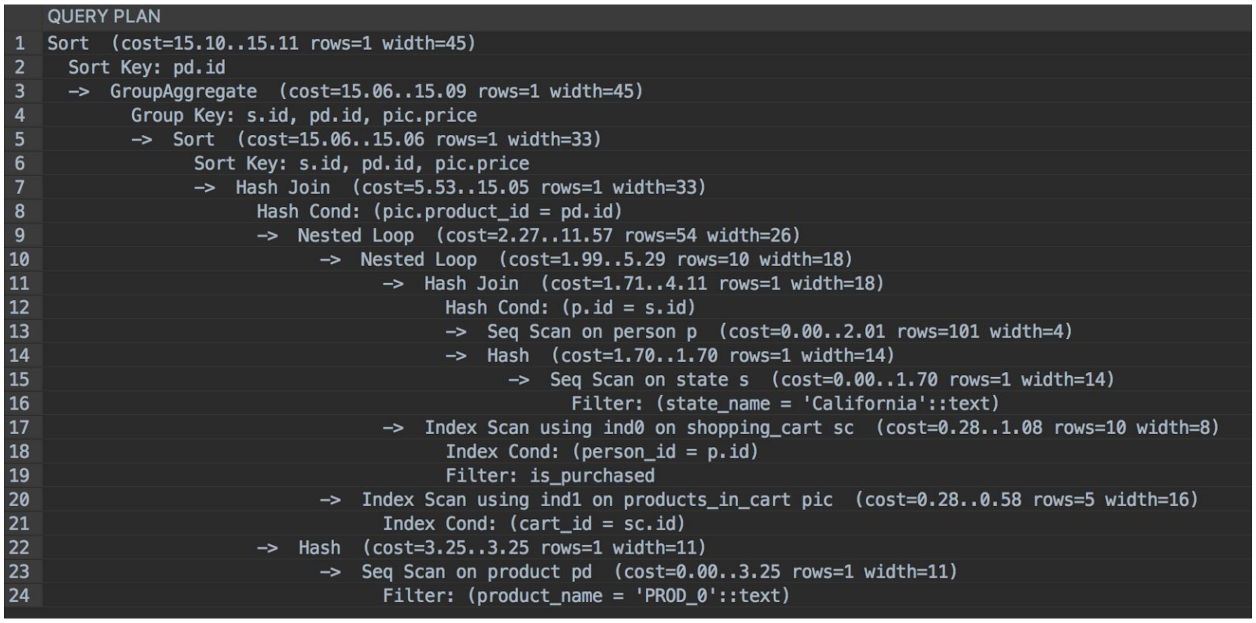
1. product(product_name)
2. products_in_cart(cart_id)
3. shopping_cart(person_id)
4. person(id) default



```
QUERY PLAN
Sort  (cost=474.36..474.37 rows=3 width=46)
  Sort Key: pd.id
  ->  GroupAggregate  (cost=474.25..474.34 rows=3 width=46)
        Group Key: s.id, pd.id, pic.price
        ->  Sort  (cost=474.25..474.26 rows=3 width=34)
              Sort Key: s.id, pd.id, pic.price
              ->  Hash Join  (cost=11.14..474.22 rows=3 width=34)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Nested Loop  (cost=2.84..455.59 rows=2747 width=26)
                          ->  Nested Loop  (cost=2.41..44.13 rows=500 width=18)
                                ->  Merge Join  (cost=1.99..4.72 rows=1 width=18)
                                      Merge Cond: (p.id = s.id)
                                      ->  Index Only Scan using person_pkey on person p  (cost=0.28..47.29 rows=1001 width=4)
                                      ->  Sort  (cost=1.71..1.72 rows=1 width=14)
                                            Sort Key: s.id
                                            ->  Seq Scan on state s  (cost=0.00..1.70 rows=1 width=14)
                                                  Filter: (state_name = 'California'::text)
                                ->  Index Scan using ind0 on shopping_cart sc  (cost=0.42..34.40 rows=500 width=8)
                                      Index Cond: (person_id = p.id)
                                      Filter: is_purchased
                          ->  Index Scan using ind1 on products_in_cart pic  (cost=0.43..0.75 rows=7 width=16)
                                Index Cond: (cart_id = sc.id)
                    ->  Hash  (cost=8.29..8.29 rows=1 width=12)
                          ->  Index Scan using ind3 on product pd  (cost=0.28..8.29 rows=1 width=12)
                                Index Cond: (product_name = 'PROD_0'::text)
```

Hot:

1. products_in_cart(cart_id) v
2. shopping_cart(person_id) v



```
QUERY PLAN
Sort  (cost=15.10..15.11 rows=1 width=45)
  Sort Key: pd.id
  ->  GroupAggregate  (cost=15.06..15.09 rows=1 width=45)
        Group Key: s.id, pd.id, pic.price
        ->  Sort  (cost=15.06..15.06 rows=1 width=33)
              Sort Key: s.id, pd.id, pic.price
              ->  Hash Join  (cost=5.53..15.05 rows=1 width=33)
                    Hash Cond: (pic.product_id = pd.id)
                    ->  Nested Loop  (cost=2.27..11.57 rows=54 width=26)
                          ->  Nested Loop  (cost=1.99..5.29 rows=10 width=18)
                                ->  Hash Join  (cost=1.71..4.11 rows=1 width=18)
                                      Hash Cond: (p.id = s.id)
                                      ->  Seq Scan on person p  (cost=0.00..2.01 rows=101 width=4)
                                      ->  Hash  (cost=1.70..1.70 rows=1 width=14)
                                            ->  Seq Scan on state s  (cost=0.00..1.70 rows=1 width=14)
                                                  Filter: (state_name = 'California'::text)
                                ->  Index Scan using ind0 on shopping_cart sc  (cost=0.28..1.08 rows=10 width=8)
                                      Index Cond: (person_id = p.id)
                                      Filter: is_purchased
                          ->  Index Scan using ind1 on products_in_cart pic  (cost=0.28..0.58 rows=5 width=16)
                                Index Cond: (cart_id = sc.id)
                    ->  Hash  (cost=3.25..3.25 rows=1 width=11)
                          ->  Seq Scan on product pd  (cost=0.00..3.25 rows=1 width=11)
                                Filter: (product_name = 'PROD_0'::text)
```