Index Writeup CSE 135

Kevin Wong
Sunny Solanki

Below lists the indexed columns that we saw gave a significant performance on our queries. To benchmark the queries we ran the queries in pgAdmin 4 and got the time with and without the indexes, with 5 trials each, and averaged them. For JSP queries we put in a timing output in the JSP java code, and it outputted the time to load the entire page when displayed. We also ran this 5 times and averaged the times to see the performance. Additionally to test between big and small data sets, we compared all the above situations with two data sets of difference sizes listed below. Although we ran these two separate data test sizes are two different computers on different operating systems with different hardware specs. It is notable to notice, that we are comparing small test results from other small test results and not small to big dataset results. The performance improves drastically on the larger database according to expected results for many of the indices chosen.

(Sunny)
Small:
100 customers
10 categories
100 products
1,000 sales

(Kevin)
Large:
1,000 customers
10 categories
1,000,000 products
500,000 sales

#1
Product Alphabetical Specified Category

Query (Get all customer alphabetically All Categories):

```
SELECT * FROM
(SELECT ROW_NUMBER() OVER(ORDER BY SUBSTRING(product_name FROM
'([0-9]+)')::BIGINT ASC, product_name) NUM, * FROM
 (
SELECT c.product_name, SUM(d.price * d.quantity), c.id
FROM product c LEFT OUTER JOIN products_in_cart d ON c.id = d.product_id JOIN category e ON
e.category_name = 'CAT_0' AND e.id = c.category_id
GROUP BY c.product_name, c.id
)  AS p
) a
WHERE NUM >= 0 AND NUM <= 20
```

Queries:

CREATE :

```
CREATE INDEX product_id
ON products_in_cart(product_id)
```

Drop:

```
Drop index product_id
```

For this query we found that indexing on the product ID offered a significant performance time improvement. In the big data set loading the JSP took 20 percent of the time of the load time of the non indexed counterpart. Noticeably **quicker** indexing on the product_id column in the products_in_cart.

Product_id	Query	JSP
Small Dataset	No index: 0.38 sec Index: 0.4 sec	No index: 399 ms Index: 369 ms
Big Dataset	No index: 3 sec. Index: 2 sec.	No index: 70978 ms Index: 13140 ms

#2
Product Alphabetical Specified Category

```
SELECT * FROM
(
 SELECT ROW_NUMBER() OVER
(ORDER BY SUBSTRING(product_name FROM '([0-9]+)')::BIGINT ASC, product_name) NUM,
* FROM (
     SELECT c.product_name,SUM(d.price * d.quantity), c.id"
       FROM product c LEFT OUTER JOIN products_in_cart d ON c.id = d.product_id
       GROUP BY c.product_name, c.id
  ) AS p ORDER BY SUBSTRING(product_name FROM '([0-9]+)')::BIGINT ASC, product_name
        ) a WHERE NUM > 0 AND NUM <= 20
```

Queries:

CREATE :

```
CREATE INDEX product_id
ON products_in_cart(product_id)
```

Drop:

```
Drop index product_id
```

CREATE :

```
CREATE INDEX price
ON products_in_cart(price)
```

Drop:

```
Drop index product_id
```

Notes: We also found a significant improvement by indexing on the product id in this similar query as well. Additionally the indexing on the price of a product offered a significant performance increase however, indexing on both of them offered slightly worse performance than either of them individually.

product_id	Query	JSP
Small Dataset	No index: 398 ms Index: 385 ms	No index: 348 ms Index: 250 ms
Big Dataset	No index:59000 ms Index: 3538 ms	No index: Index:

price	Query	JSP

Small Dataset	No index: 392 ms Index: 404 ms	No index: 328 ms Index: 359 ms
Big Dataset	No index:59000 ms Index: 3538 ms	No index: 63000 Index: 5818

#3 Customer Alphabetical All Categories & Specified Category

```
SELECT * FROM  (
     SELECT ROW_NUMBER() OVER(ORDER BY SUBSTRING(person_name FROM '([0-9]+)')::BIGINT
ASC, person_name) NUM, * FROM
       (
         SELECT c.person_name, SUM(e.price * e.quantity)
         FROM person c LEFT OUTER JOIN shopping_cart d ON c.id = d.person_id JOIN
           products_in_cart e ON d.id = e.cart_id AND d.is_purchased = true
          GROUP BY c.person_name
        ) AS p ) a
        WHERE NUM > 0 AND NUM <= 20

SELECT * FROM" +
       " (" +
       " SELECT ROW_NUMBER() OVER(ORDER BY SUBSTRING(person_name FROM
'([0-9]+)')::BIGINT ASC, person_name) NUM, * FROM" +
       " (" +
       "   SELECT c.person_name, SUM(e.price * e.quantity)" +
       "   FROM person c LEFT OUTER JOIN shopping_cart d ON c.id = d.person_id JOIN
products_in_cart e ON d.id = e.cart_id AND d.is_purchased = true" +
       "   JOIN Category f on f.category_name = ? JOIN Product g ON g.category_id =
f.id AND g.id = e.product_id " +
       "   GROUP BY c.person_name" +
       " ) AS p ) a " +
       " WHERE NUM > ? AND NUM <= ?"
```

We also noticed throughout the testing, the queries which were similar, in that they searched for alphabetical customers and products, that the product id index offered significant performance increases throughout the queries. We speculate this is the case due to the similarity of the queries.

#4 Total Sales Alphabetical All Categories & Specified Category

```
SELECT SUM(c.price * c.quantity) AS totalSales
FROM person a, shopping_cart b, products_in_cart c, product d
WHERE a.person_name = ‘CUST_0’ AND b.person_id = a.id AND c.cart_id = b.id AND
c.product_id = d.id AND d.id = 4
```

```
SELECT SUM(c.price * c.quantity) as totalSales
FROM person a, shopping_cart b, products_in_cart c, product d, category e
 WHERE a.person_name = ‘CUST_0’ AND b.person_id = a.id AND c.cart_id = b.id AND
c.product_id = d.id
 AND d.id = 4 AND e.category_name = ‘CAT_2’ AND d.category_id = e.id
```

CREATE :

```
CREATE INDEX product_category_id
ON product(category_id)
```

We found that product_category_id performed slightly better on the smaller data set, while strangely, it did not perform significantly better on the larger data set, on either JSP or in the query for PgAdmin. We also tested on person id and cart id, which both did not offer much of a difference in terms of performance.

category_id	Query	JSP
Small Dataset	No index:403 ms Index: 413 ms	No index: 409 ms Index: 358 ms
Large Dataset	No index: 120 ms Index: 109 ms	No index: 360 ms Index: 365 ms

#5
States Alphabetical All Categories & Specified Category

```
SELECT * FROM
        (
         SELECT ROW_NUMBER() OVER(ORDER BY SUBSTRING(state_name FROM
'([0-9]+)')::BIGINT ASC, state_name) NUM, * FROM
       (
         SELECT f.state_name, f.id ,SUM(e.price * e.quantity)
        FROM person c LEFT OUTER JOIN shopping_cart d ON c.id = d.person_id JOIN
products_in_cart e ON d.id = e.cart_id
         AND d.is_purchased = true JOIN state f ON f.id = c.state_id
        GROUP BY f.state_name, f.id
        ) AS p ) a
        WHERE NUM > 0 AND NUM <= 20

SELECT * FROM
 (
   SELECT ROW_NUMBER() OVER(ORDER BY SUBSTRING(state_name FROM
'([0-9]+)')::BIGINT ASC, state_name) NUM, * FROM
   (
    SELECT f.state_name, f.id ,SUM(e.price * e.quantity)
    FROM person c LEFT OUTER JOIN shopping_cart d ON c.id = d.person_id JOIN
products_in_cart e ON d.id = e.cart_id
    AND d.is_purchased = true JOIN state f ON f.id = c.state_id JOIN Category g ON
g.category_name = ‘CAT_0’
    JOIN product h ON h.category_id = g.id AND h.id = e.product_id
    GROUP BY f.state_name, f.id
     ) AS p ) a
    WHERE NUM > 0 AND NUM <= 20
```

CREATE :

```
CREATE INDEX person_id
ON shopping_cart(person_id)
```

Note:

We noticed that indexing on the person_id gave slightly better performance for smaller data sets in both the JSP load times and querying. However, this same trend was not shown in the larger data set, in fact it performed worse than having no index for both querying and loading the JSP page. Additionally, indexing on product_id offered almost a 4 times performance improvement, when not being indexed at all.

	Query	JSP
Small Dataset	No index: 450 ms Index: 320 ms	No index: 396 ms Index: 260 ms
Large Dataset	No index: 3 sec Index: 3 sec	No index: 11500 ms Index: 12690 ms

6

Total Sales for Alphabetical States All Categories & Specified Category

```
SELECT SUM(c.price * c.quantity) as totalSales
 FROM person a, shopping_cart b, products_in_cart c, product d, category e, state f
 WHERE f.id = 2 AND a.state_id = f.id AND b.person_id = a.id AND c.cart_id = b.id AND
c.product_id = d.id
AND d.id = 4 AND e.category_name = ‘CAT_0’ AND d.category_id = e.id
```

```
SELECT SUM(c.price * c.quantity) AS totalSales
 FROM person a, shopping_cart b, products_in_cart c, product d, state e
 WHERE e.id = 2 AND a.state_id = e.id AND b.person_id = a.id AND c.cart_id = b.id AND
c.product_id = d.id AND d.id = 4
```

Note: We tested indexing on category id and person id, and neither of them offered an improvement and in most cases made the performance worse.

CREATE :

```
CREATE INDEX person_id
ON shopping_cart(person_id)
```

#7

Top Products for Top Customers All Categories & Specified Category

```
SELECT * FROM ( SELECT ROW_NUMBER() OVER (ORDER BY totalSales DESC) NUM,  *
FROM  (SELECT a.product_id, c.product_name, SUM(a.price * a.quantity) as totalSales
FROM products_in_cart a, shopping_cart b, product c
WHERE a.cart_id = b.id AND b.is_purchased = true AND c.id = a.product_id
GROUP BY c.product_name, a.product_id ) as S
) as topProducts
WHERE NUM > 0 AND NUM <= 20
ORDER BY totalSales DESC
```

```
SELECT * FROM ( SELECT ROW_NUMBER() OVER (ORDER BY totalSales DESC) NUM,
* FROM (SELECT a.product_id, c.product_name, d.category_name, SUM(a.price * a.quantity)
as totalSales
FROM products_in_cart a, shopping_cart b, product c, category d
WHERE a.cart_id = b.id AND b.is_purchased = true AND c.id = a.product_id AND c.category_id
= d.id AND d.category_name = ‘CAT_0’
GROUP BY a.product_id, c.product_name, d.category_name
) as S ) as topProducts WHERE NUM >0 AND NUM <= 20
ORDER BY totalSales DESC
```

```
CREATE INDEX cart_id
ON products_in_cart(cart_id)
```

Noticed significant improvements for the small and large data set. Particularly the large dataset when loading the JSP page.

	Query	JSP
Small Dataset	No index: 440 ms Index: 365 ms	No index: 430 ms Index: 282 ms
Large Dataset	No index: 16 s Index: 15 s	No index: 75000 ms Index: 30000 ms

```
CREATE INDEX product_id
ON products_in_cart(product_id)
```

For both product_id index and cart_id dataset we noticed both having a sizeable performance increse with the most notable with the JSP page load times of the large dataset.

	Query	JSP
Small Dataset	No index: 429 ms Index: 371 ms	No index: 467 ms Index: 219 ms
Large Dataset	No index: 16 s Index: 15 s	No index: 77000 ms Index: 15000 ms

#8

Top Products for Top States All Categories & Specified Category

```
SELECT * FROM ( SELECT ROW_NUMBER() OVER (ORDER BY totalSales DESC) NUM,  *
FROM  (SELECT a.product_id, c.product_name, SUM(a.price * a.quantity) as totalSales
FROM products_in_cart a, shopping_cart b, product c
WHERE a.cart_id = b.id AND b.is_purchased = true AND c.id = a.product_id
GROUP BY c.product_name, a.product_id ) as S
  ) as topProducts
     WHERE NUM >=0 AND NUM <= 20
    ORDER BY totalSales DESC
```

```
SELECT * FROM ( SELECT ROW_NUMBER() OVER (ORDER BY totalSales DESC) NUM,
  * FROM (SELECT a.product_id, c.product_name, d.category_name, SUM(a.price * a.quantity)
as totalSales
  FROM products_in_cart a, shopping_cart b, product c, category d
  WHERE a.cart_id = b.id AND b.is_purchased = true AND c.id = a.product_id AND
c.category_id = d.id AND d.category_name = ‘CAT_0’
  GROUP BY a.product_id, c.product_name, d.category_name
  ) as S ) as topProducts WHERE NUM >=0 AND NUM <=20
  ORDER BY totalSales DESC
```

```
CREATE INDEX product_id
ON products_in_cart(product_id)
```

	Query	JSP
Small Dataset	No index: 419 ms Index: 384 ms	No index: 467 ms Index: 219 ms
Large Dataset	No index: 16 s Index: 15 s	No index: 76050 ms Index: 17420 ms

```
CREATE INDEX category_id
ON category(id)
```

	Query	JSP
Small Dataset	No index: 440 ms Index: 345 ms	No index: 430 ms Index: 240 ms
Large Dataset	No index:16s Index: 15 s	No index: 67030 ms Index:17003 ms

Overall, throughout our tests we noticed a trend of indexing on product id and category id having the biggest effect on performance. We speculate that indexing both would lead to the most optimal performance. The reason that the product id and category id indexes have such a big impact on performance is because for every query to fill the tables, there is a check for a matching category id and product id between tables.