Indices' Report

Possible useful indices:

products_in_cart(cart_id);
products_in_cart (product_id);
shopping_cart (person_id);
shopping_cart (is_purchased);
product (category_id);
person (state_id);
state (state_code);

Reason for choosing those indices:

We Think indices may help improve the performance of the query only when it's created for the columns we used for join condition and in ORDER clause.
Since the id columns in every tables have indices created automatically, we don't need to create duplicate ones for them. So, we think creating indices for the rest columns used in the join condition and ORDER clauses is most promising on improving the query performance.

We picked products_in_cart(cart_id) as our index because when in all 8 cases of our queries, we need to join shopping_cart table and products_in_cart table on the condition that product_in_cart.cart_id equals shopping_cart.id. Using products_in_cart(cart_id) as an index will let the query quickly obtain desired shopping_cart tuples. This index is really helpful in both row queries and cell queries.

We picked products_in_cart(product_id) as our index because of similar reasons as above. When joining product table and products_in_cart table on the condition that product.id equals products_in_cart.product_id, the index on product_id will help the query obtain the matching product tuples efficiently. This index is very helpful in both column queries and cell queries.

shopping_cart(person_id) is also a helpful index in joining shopping_cart table and person table on the condition that shopping_cart.person_id equals person.id. This index will be helpful in all three queries.

shopping_cart(is_purchased) is a index in separating the rows of shopping cart that we want to sum and the rows of shopping cart that we don't want to sum up. This index may be less helpful since it only contains two values which is true and false.

person(state_id) is used to join the person table with the state table when we choose to group analysis results by states instead of the customers. This index may only help in generating the get row queries.

state(state_code) is used in the same way as the previous person(state_id). We think this one and person(state_id) can be interchangeable and has the same helpful level.

product(category_id) will only become helpful in the case that filter option is at one specified category. In the nested queries to get all products in the targeted category, this index will help obtain desired product tuples quickly.

Above are our reasonings for picking selected indices. However, in real-time situations, in some cases, not all of the indices will be actually used by postgresql. Below are our experiment data on selected indices.

Also, since primary keys are defaulted to be indices in postgresql. In our experimentation, we find the following ids are used in our queries: person(id), shopping_cart(id), product(id).

Reference to table size:

SMALL TABLE:	LARGE TABLE:
25 customers	1000 customers
30 products	1000000 products
10 categories	10 categories
80 sales	500000 sales

Experiment data:

ALL CATEGORY:

CASE	index	SMALL TABLE	LARGE TABLE
C ALPHA ALL	pic_product_id shopping_cart_pkey	doesn't work	work
C ALPHA ALL	pic_cart_id	doesn't work	work
C ALPHA ALL	shopping_cart_pkey	doesn't work	work
C TOPK ALL	pic_product_id shopping_cart_pkey	doesn't work	work
C TOPK ALL	pic_cart_id	doesn't work	work
C TOPK ALL	shopping_cart_pkey	doesn't work	work
S ALPHA ALL	pic_product_id shopping_cart_pkey	doesn't work	work
S ALPHA ALL	shopping_cart_pkey person_pkey	doesn't work	work
S TOPK ALL	pic_product_id shopping_cart_pkey	doesn't work	work
S TOPK ALL	shopping_cart_pkey person_pkey	doesn't work	work

Test C ALPHA ALL:

1. Without indexing

	SMALL	LARGE
Cost	86.07	889806.26
Query time 1	150ms	11s
Query time 2	148ms	11s
Query time 3	154ms	11s
JSP time 1	12ms	18839ms
JSP time 2	13ms	19397ms
JSP time 3	14ms	18637ms

2. With indexing (pic_product_id)

	SMALL	LARGE
Cost	-	834682.3
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	18093ms
JSP time 2	-	17618ms
JSP time 3	-	17976ms

3. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	-	855424.16
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	18040ms
JSP time 2	-	17676ms
JSP time 3	-	17053ms

Test C TOPK ALL:

4. Without indexing

	SMALL	LARGE
Cost	86.07	889806.26
Query time 1	143ms	11s
Query time 2	152ms	11s
Query time 3	159ms	11s
JSP time 1	9ms	18442ms
JSP time 2	10ms	19739ms
JSP time 3	9ms	18876ms

5. With indexing (pic_product_id)

	SMALL	LARGE
Cost	-	834682.3
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	18277ms
JSP time 2	-	17744ms
JSP time 3	-	17898ms

6. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	-	855424.16
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	18225ms
JSP time 2	-	17123ms
JSP time 3	-	17893ms

Test S ALPHA ALL:

7. Without indexing

	SMALL	LARGE
Cost	102.96	890771.17
Query time 1	139	12s
Query time 2	155	12s
Query time 3	144	12s
JSP time 1	10	18578ms
JSP time 2	9	19064ms
JSP time 3	9	18921ms

8. With indexing (pic_product_id)

	SMALL	LARGE
Cost	-	835653.71
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	17640ms
JSP time 2	-	17843ms
JSP time 3	-	18027ms

Test S TOPK ALL:

9. Without indexing

	SMALL	LARGE
Cost	102.96	890771.17
Query time 1	144	11s
Query time 2	150	11s
Query time 3	154	11s
JSP time 1	9	18455ms
JSP time 2	8	19498ms

JSP time 3	8	18858ms

10. With indexing (pic_product_id)

	SMALL	LARGE
Cost	-	835653.71
Query time 1	-	10s
Query time 2	-	10s
Query time 3	-	10s
JSP time 1	-	17554ms
JSP time 2	-	17921ms
JSP time 3	-	18007ms

ONE CATEGORY:

CASE	index	SMALL TABLE	LARGE TABLE
C ALPHA ONE	sc_person_id, pic_product_id, prod_category_id	works	works
C ALPHA ONE	pic_cart_id	works	works
C TOPK ONE	sc_person_id, pic_product_id, prod_category_id	works	works
C TOPK ONE	pic_cart_id	works	works
S ALPHA ONE	sc_person_id, pic_product_id, prod_category_id	works	works
S ALPHA ONE	pic_cart_id	works	doesn't work
S TOPK ONE	sc_person_id, pic_product_id, prod_category_id	works	works
S TOPK ONE	pic_cart_id	works	doesn't work

Test C ALPHA ONE:

1. Without indexing

	SMALL	LARGE
Cost	41.01	384327.51
Query time 1	80ms	4s
Query time 2	75ms	4s
Query time 3	74ms	4s
JSP time 1	9ms	10431ms
JSP time 2	8ms	10439ms
JSP time 3	9ms	10459ms

2. With indexing (sc_person_id, pic_product_id, prod_category_id)

	SMALL	LARGE
Cost	14.78	229380.83
Query time 1	73ms	3s
Query time 2	73ms	3s
Query time 3	68ms	3s
JSP time 1	8ms	9004ms
JSP time 2	8ms	9000ms
JSP time 3	7ms	9106ms

3. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	40.86	347811.81
Query time 1	72ms	4s
Query time 2	74ms	4s
Query time 3	72ms	4s
JSP time 1	9ms	10128ms
JSP time 2	9ms	9557ms
JSP time 3	9ms	10341ms

Test C TOPK ONE:

1. Without indexing

	SMALL	LARGE
Cost	41.01	384327.51
Query time 1	72ms	4s
Query time 2	73ms	4s
Query time 3	73ms	4s
JSP time 1	7ms	10125ms
JSP time 2	7ms	10501ms
JSP time 3	6ms	10101ms

2. With indexing (sc_person_id, pic_product_id, prod_category_id)

	SMALL	LARGE
Cost	14.78	229380.83
Query time 1	72ms	3s
Query time 2	76ms	3s
Query time 3	72ms	3s
JSP time 1	8ms	9353ms
JSP time 2	7ms	9308ms
JSP time 3	7ms	8849ms

3. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	40.86	347811.81
Query time 1	80ms	4s
Query time 2	74ms	4s
Query time 3	76ms	4s
JSP time 1	9ms	10234ms
JSP time 2	6ms	9975ms
JSP time 3	8ms	9567ms

Test S ALPHA ONE:

1. Without indexing

	SMALL	LARGE
Cost	46.41	384310.5
Query time 1	76ms	4s
Query time 2	77ms	4s
Query time 3	74ms	4s
JSP time 1	8ms	9944ms
JSP time 2	7ms	9904ms
JSP time 3	7ms	9850ms

2. With indexing (sc_person_id, pic_product_id, prod_category_id)

	SMALL	LARGE
Cost	19.14	229374.93
Query time 1	73ms	3s
Query time 2	71ms	3s
Query time 3	76ms	3s
JSP time 1	7ms	9494ms
JSP time 2	7ms	9078ms
JSP time 3	7ms	9109ms

3. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	45.26	-
Query time 1	76ms	-
Query time 2	74ms	-
Query time 3	75ms	-
JSP time 1	9ms	-
JSP time 2	6ms	-
JSP time 3	7ms	-

Test S TOPK ONE:

1. Without indexing

	SMALL	LARGE
Cost	46.41	384310.5
Query time 1	76ms	4s
Query time 2	75ms	4s
Query time 3	73ms	4s
JSP time 1	8ms	10468ms
JSP time 2	7ms	7146ms
JSP time 3	6ms	7890ms

2. With indexing (sc_person_id, pic_product_id, prod_category_id)

	SMALL	LARGE
Cost	19.14	229374.93
Query time 1	76ms	3s
Query time 2	75ms	3s
Query time 3	74ms	3s
JSP time 1	7ms	6810ms
JSP time 2	8ms	6402ms
JSP time 3	8ms	9083ms

3. With indexing (pic_cart_id)

	SMALL	LARGE
Cost	45.28	-
Query time 1	74ms	-
Query time 2	75ms	-
Query time 3	72ms	-
JSP time 1	7ms	-
JSP time 2	9ms	-
JSP time 3	7ms	-

Conclution:

According to our experiment, there are only four indicing that are finally used by the sql database: sc_person_id, pic_product_id, pic_cart_id and prod_category_id. The rest ‘may help’ indices are actually will cost more than querying without them and so they are not used.

We also noticed that in some case, for example, when we want to search customers’ shopping summary for all the categories and the datatable is small, the sql will decide to use the sequential indexing way rather than the one with indices. In general, in a table that is small and hot, creating all four indices doesn’t speed things up dramatically and not all of them will be used. On the contrary, for a table that is large and cold, creating all four indices will definitely improve performance, especially in the case that one specified category is selected.