EPAM Systems, RD Dep., RD Dep.

POSTGRESQL DB FOR DWH AND ETL BUILDING

PostgreSQL Join Methods

CONTENTS

1. JOIN METHODS

Read about PostgreSQL parameters enable_nestloop, enable_hashjoin, enable_mergejoin and how they can be used to instruct the planner to choose a join method.

Task Results: Provide queries where needed. Read the plan and describe what happened and why with screenshots where needed.

Performing tasks below you can modify not only queries but tables whatever you want: add, delete rows, indexes and etc.

1.1. TASK 1: NESTED LOOP JOIN

1. Create test tables and populate them with test data:

```
CREATE TABLE test_joins_a
(
id1 int,
id2 int
);

CREATE TABLE test_joins_b
(
id1 int,
id2 int
);

INSERT INTO test_joins_a values(generate_series(1,10000),3);

INSERT INTO test_joins_b values(generate_series(1,10000),3);

ANALYZE;
```

2. Check how NESTED LOOP JOIN method is used in below queries. Why?

```
SELECT * FROM test_joins_a a, test_joins_b b
WHERE a.id1 > b.id1;

SELECT *
FROM test_joins_a a
CROSS JOIN test_joins_b b;
```

1.2 TASK 2: HASH JOIN

1. **Rewrite SELECT to instruct the planner to use HASH JOIN method:**

```
SELECT * FROM test_joins_a a, test_joins_b b
WHERE a.id1 > b.id1;
```

2. **Create query with SEMI JOIN between tables to get HASH SEMI JOIN in the plan.**
3. Set enable_hashjoin to off and recheck plan. Switch on enable_hashjoin.

1.3 TASK 3: MERGE JOIN

1. **Using tables** test_joins_a and test_joins_b create a query which is use MERGE JOIN as a join method.
2. Set enable_mergejoin to off and recheck plan. Switch on enable_mergejoin.

2. JOIN ORDER AND LATERAL JOIN

2.1 TASK 4: CHANGING JOIN ORDER

1. Create a table and populate it with sample data:

```
CREATE TABLE test_joins_c
(
id1 int,
id2 int
);


INSERT INTO test_joins_c
     values(generate_series(1,1000000),(random()*10)::int);
```

2. Check the plan. Describe the order of tables joining:

```
EXPLAIN
SELECT c.id2
FROM test_joins_b b
JOIN test_joins_a a on (b.id1 = a.id1)
LEFT JOIN test_joins_c c on (c.id1 = b.id1);
```

3. Set join_collapse_limit = 1 and recreate plan for query above. Describe changes if any. Return join_collapse_limit = 8.

2.2 TASK 5: LATERAL JOIN

1. Create tables and populate them by data:

```
CREATE TABLE orders AS
SELECT   id AS order_id,
             (id * 10 * random()*10)::int AS order_cost,
             'order number ' || id AS order_num
FROM generate_series(1, 1000) AS id;


CREATE TABLE stores (
     store_id int,
     store_name text,
     max_order_cost int
);
```

```
INSERT INTO stores VALUES
    (1, 'grossery shop', '800'),
    (2, 'bakery', '100'),
    (3, 'manufactured goods', '3000')
;
```

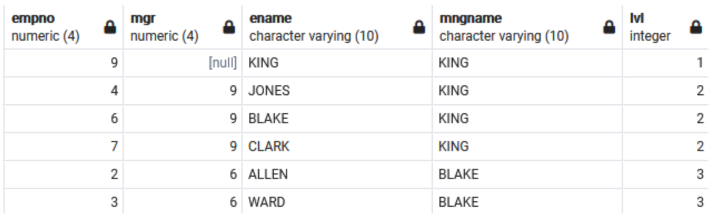
2. Create a query to find TOP 10 of orders by it cost for each store. So, on the output you should have 10 orders for each store (or less, depends on sample random data) with cost less than max_order_cost. Use LATERAL join.

3. CTES

3.1 TASK 6: RECURSIVE CTE

1. Use emp table you created before. Select all employee and his manager name and level of management start from president of the company

Example:



empno numeric (4)	mgr numeric (4)	ename character varying (10)	mngname character varying (10)	lvl integer
9	[null]	KING	KING	1
4	9	JONES	KING	2
6	9	BLAKE	KING	2
7	9	CLARK	KING	2
2	6	ALLEN	BLAKE	3
3	6	WARD	BLAKE	3

3.2 TASK 7: CHANGING DATA CTE

1. Create log table for emp table:

```
CREATE TABLE order_log
(
   log_id       integer primary key generated always as identity,
   order_id     integer,
   order_cost   integer,
   order_num    text,
   action_type  varchar(1) CHECK (action_type IN ('U','D')),
   log_date     TIMESTAMPTZ DEFAULT Now()
);
```

2. Update all rows for ORDER table:
   a. set new ORDER_COST = (old ORDER_COST / 2) where old ORDER_COST between 100 and 1000
   b. delete all rows where ORDER_COST < 50
   c. save all updated and deleted rows into log table with action type 'U' and 'D' relatively.

   Perform all in one SQL CTE query.