第 9 章 查询处理实验报告

3022005055 李燚坡 人工智能 4 班

启动服务器，drop 之前的表并建立新表。
启动报错 failed to connect，用 v3.0.0]$ gs_ctl start -D $GAUSSHOME/data -Z single_node-l logfile 解决
Drop user 失败，使用 DROP TABLE users CASCADE;
并成功创建表。

```
constraint fk_orders_station_end on table orders1 depends on table station
constraint fk_orders_station_start on table orders2 depends on table station
constraint fk_orders_station_end on table orders2 depends on table station
constraint fk_trainstop_station on table trainstop2 depends on table station
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
ERROR:  cannot drop table users because other objects depend on it
DETAIL:  constraint fk_orders_users on table orders1 depends on table users
constraint fk_orders_users on table orders2 depends on table users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
DROP TABLE
railway=# clear
railway-#
railway-#
railway-#
railway-# ;
ERROR:  syntax error at or near "clear"
LINE 1: clear
        ^

railway=# CREATE TABLE users
railway-# (
railway(# u_id varchar(20),       -- 用户 id，用于系统登录账户名（主键）
railway(# u_passwd varchar(20),     -- 密码，用于系统登录密码
railway(# u_name varchar(10),      -- 真实姓名
railway(# u_idnum varchar(20),      -- 证件号码
railway(# u_regtime timestamp,      -- 注册时间
railway(#
railway(#
railway(#
railway(#
railway(#
railway(# CONSTRAINT pk_users PRIMARY KEY (u_id)
railway(# );
ERROR:  relation "users" already exists in schema "public"
DETAIL:  creating new table with existing name in the same schema
railway=# DROP TABLE users;
ERROR:  cannot drop table users because other objects depend on it
DETAIL:  constraint fk_orders_users on table orders1 depends on table users
constraint fk_orders_users on table orders2 depends on table users
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
railway=# DROP TABLE users CASCADE;
NOTICE:  drop cascades to 2 other objects
DETAIL:  drop cascades to constraint fk_orders_users on table orders1
drop cascades to constraint fk_orders_users on table orders2
DROP TABLE
railway=# CREATE TABLE users
(
u_id varchar(20),       -- 用户 id，用于系统登录账户名（主键）
u_passwd varchar(20),     -- 密码，用于系统登录密码
u_name varchar(10),      -- 真实姓名
u_idnum varchar(20),      -- 证件号码
u_regtime timestamp,      -- 注册时间
CONSTRAINT pk_users PRIMARY KEY (u_id)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "pk_users" for table
"users"
CREATE TABLE
railway=#
```

```
railway=# drop TABLE orders;
DROP TABLE
railway=# CREATE TABLE orders
railway-# (
railway(# o_id int,
railway(# o_uid varchar(20),
railway(# o_tdate date,
railway(# o_tid varchar(10),
railway(# o_sstation varchar(20),
railway(# o_estation varchar(20),
railway(# o_seattype smallint,
railway(# o_carriage smallint,
railway(# o_seatnum smallint,      -- 座位号（排）
railway(# o_seatloc char(1),       -- 座位位置：ABCEF
railway(# o_price money,          -- 订单金额
railway(# o_ispaid boolean,        -- 是否已支付
railway(# o_ctime timestamp,       -- 订单创建时间
railway(# CONSTRAINT pk_orders PRIMARY KEY (o_id)
railway(# );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "pk_orders" for table
 "orders"
CREATE TABLE
railway=# INSERT INTO orders VALUES(1,'1','2022-04-29','C2002','天津','北京南',2,8,7
,'F',54,1,'2022-04-27 16:00:12'); INSERT INTO orders VALUES(2,'3','2022-04-29','G321
','天津南','福',1,4,7,'A',742.5,1,'2022-04-27 17:00:12'); INSERT INTO orders VALUES(
3,'3','2022-04-29','G1709','天津西','重庆',2,9,3,'D',929,1,'2022-04-27 18:00:12');
INSERT 0 1
INSERT 0 1
INSERT 0 1
railway=# SELECT u_id, u_name FROM users;
 u_id | u_name
------+--------
 1    | 张三
 2    | 李四
 3    | 王五
(3 rows)

railway=# SELECT o_id, o_uid, o_tid FROM orders;
 o_id | o_uid | o_tid
------+-------+-------
    1 | 1     | C2002
    2 | 3     | G321
    3 | 3     | G1709
(3 rows)

railway=#
```

```
[dblab@ecs-541a openGauss-server-v3.0.0]$ gs_ctl start -D $GAUSSHOME/data -Z single_node -l logfile
[2024-04-15 15:18:48.059][194872][][gs_ctl]: gs_ctl started,datadir is /home/dblab/opengauss-compile/o
penGauss-server-v3.0.0/dest/data
[2024-04-15 15:18:48.099][194872][][gs_ctl]: waiting for server to start...
.
[2024-04-15 15:18:49.113][194872][][gs_ctl]:  done
[2024-04-15 15:18:49.113][194872][][gs_ctl]: server started (/home/dblab/opengauss-compile/openGauss-s
erver-v3.0.0/dest/data)
[dblab@ecs-541a openGauss-server-v3.0.0]$ gsql railway -r
gsql ((openGauss 3.0.0 build ) compiled at 2024-03-26 19:20:32 commit 0 last mr  debug)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    3 | 3     | G1709
 3    | 王五   |    2 | 3     | G321
(3 rows)

railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid
;
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=16.46..33.82 rows=287 width=196)
   Hash Cond: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Seq Scan on users  (cost=0.00..13.27 rows=327 width=96)
   ->  Hash  (cost=12.87..12.87 rows=287 width=100)
         ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
(5 rows)

railway=# SET enable_hashjoin = off;
SET
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)

railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid
;
                              QUERY PLAN
-----------------------------------------------------------------------
 Merge Join  (cost=51.51..57.45 rows=287 width=196)
   Merge Cond: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Sort  (cost=26.93..27.74 rows=327 width=96)
         Sort Key: users.u_id
         ->  Seq Scan on users  (cost=0.00..13.27 rows=327 width=96)
   ->  Sort  (cost=24.59..25.30 rows=287 width=100)
         Sort Key: orders.o_uid
         ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
(8 rows)

railway=#
```

```
railway=# SET enable_mergejoin = off;
SET
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: no outer tuple, ending join
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)

railway=#
```

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid
;
                                QUERY PLAN
----------------------------------------------------------------------------
 Nested Loop  (cost=0.00..141.23 rows=287 width=196)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Index Scan using pk_users on users  (cost=0.00..0.44 rows=1 width=96)
         Index Cond: ((u_id)::text = (orders.o_uid)::text)
(4 rows)

railway=#
```

```
[dblab@ecs-541a openGauss-server-v3.0.0]$ gsql railway -r
gsql ((openGauss 3.0.0 build ) compiled at 2024-03-26 19:20:32 commit 0
last mr  debug)
Non-SSL connection (SSL connection is recommended when requiring high-se
curity)
Type "help" for help.

railway=# SET enable_hashjoin = off;
SET
railway=# SET enable_mergejoin = off;
SET
railway=#  SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN
 orders ON u_id = o_uid;
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 1 < -- > inner attr: u_id = 1
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 3
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 3
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: no outer tuple, ending join
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)

railway=#
```

```
railway=# SET enable_indexscan = off;
SET
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INN
ER JOIN orders ON u_id = o_uid;
                                QUERY PLAN
------------------------------------------------------------------------
-----
 Nested Loop  (cost=0.29..1249.30 rows=287 width=196)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Bitmap Heap Scan on users  (cost=0.29..4.30 rows=1 width=96)
         Recheck Cond: ((u_id)::text = (orders.o_uid)::text)
         ->  Bitmap Index Scan on pk_users  (cost=0.00..0.29 rows=1 widt
h=0)
               Index Cond: ((u_id)::text = (orders.o_uid)::text)
(6 rows)

railway=# SET enable_material = off;
SET
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INN
ER JOIN orders ON u_id = o_uid;
                                QUERY PLAN
------------------------------------------------------------------------
-----
 Nested Loop  (cost=0.29..1249.30 rows=287 width=196)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Bitmap Heap Scan on users  (cost=0.29..4.30 rows=1 width=96)
         Recheck Cond: ((u_id)::text = (orders.o_uid)::text)
         ->  Bitmap Index Scan on pk_users  (cost=0.00..0.29 rows=1 widt
h=0)
               Index Cond: ((u_id)::text = (orders.o_uid)::text)
(6 rows)

railway=# SET enable_bitmapscan = off;
SET
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INN
ER JOIN orders ON u_id = o_uid;
                           QUERY PLAN
-----------------------------------------------------------------
 Nested Loop  (cost=0.00..4994.47 rows=287 width=196)
   Join Filter: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Seq Scan on users  (cost=0.00..13.27 rows=327 width=96)
(4 rows)

railway=#
```

```
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 1 < -- > inner attr: u_id = 2
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 1 < -- > inner attr: u_id = 3
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 1
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 2
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 3
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 1
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 2
INFO:  ExecNestLoop: qualification failed, looping
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  outer attr: o_uid = 3 < -- > inner attr: u_id = 3
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: no outer tuple, ending join
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)

railway=#
```

在实验步骤 9.4.4 节中，连接属性值比较信息输出的行数变化是由于查询执行计划的变化引起的。

1. 配置前的执行计划：在没有禁用`enable_indexscan`、`enable_material`和`enable_bitmapscan`之前，查询执行计划利用了索引扫描(Index Scan)。由于`users`表的`u_id`和`orders`表的`o_uid`之间使用了索引进行比较，每个`orders`表的元组都直接通过索引找到了与之匹配的`users`表中的唯一元组。因此，每个`orders`表的元组仅需进行一次比较就能找到匹配项。由于示例中只有 3 个订单记录，并且它们与用户记录一一对应，所以总共产生了 3 行输出。

2.配置后的执行计划：在将`enable_indexscan`、`enable_material`和`enable_bitmapscan`配置为`off`之后，查询执行计划改变为全表扫描(Seq Scan)加嵌套循环(Nested Loop)。在此情况下，系统没有利用索引，而是采用了更基础的方法：对于`users`表中的每一行，系统会扫描`orders`表中的所有行以寻找匹配项。因此，每一对用户和订单的可能组合都被比较了，即使多数比较并不成功。在给定的数据中，由于每个用户都被尝试匹配到每个订单，因此生成了更多的比较输出，总共 9 行。

这种变化说明禁用索引和物化视图等特定扫描类型会导致数据库查询优化器采用不同的策略来执行查询，这可能会影响查询的效率和输出的详细程度。在实验中，这种改变有助于演示和验证嵌套循环算法的工作原理，尤其是在不同的扫描策略之间转换时。