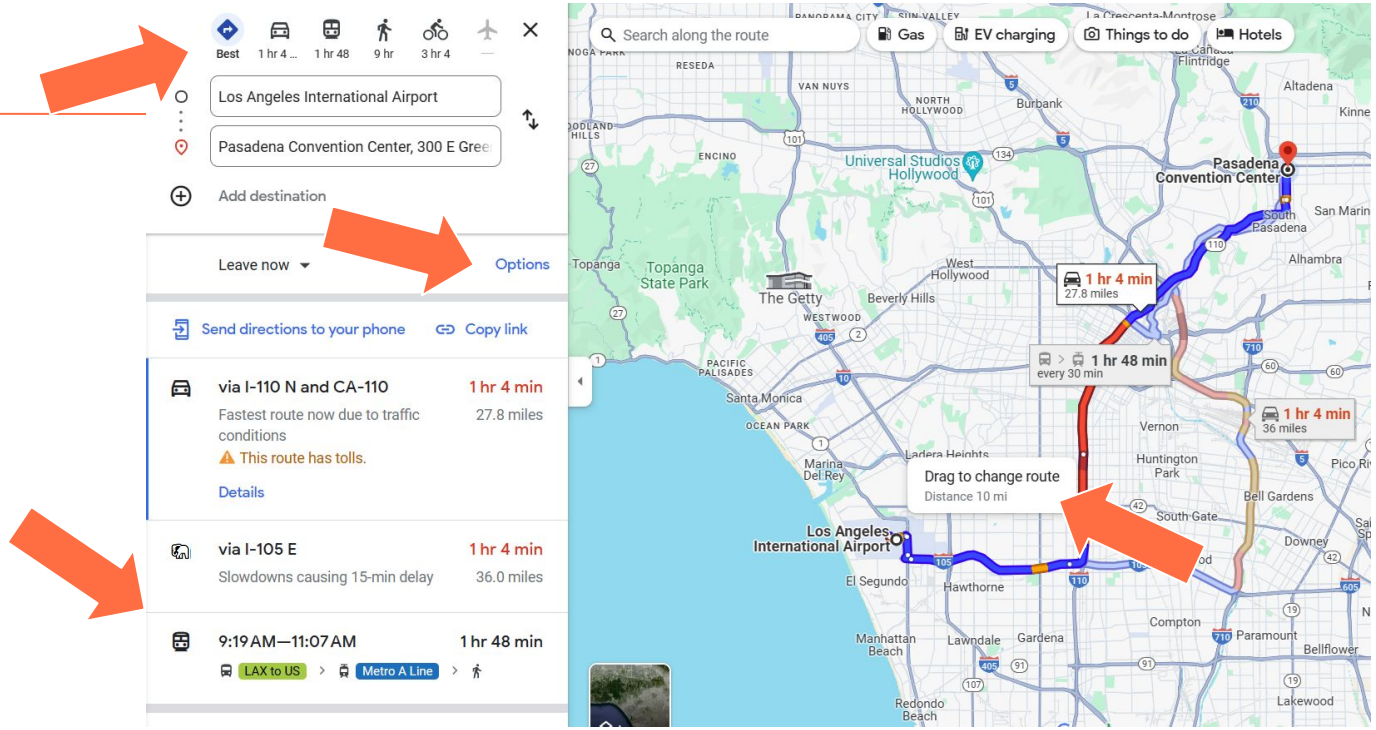
pg_hint_plan

get the right plan without surprises

Amit Chauhan
Principal Solutions Engineer

yugabyteDB

Hints... why?



Would it make sense to provide Google Maps without the choice of travel mode, route options, drag to change route?

Hints... why in SQL?

SQL is a declarative language

the query planner generates the procedural code to access data

👉 You may want to understand its choices

👉 You may want to workaround bad choices

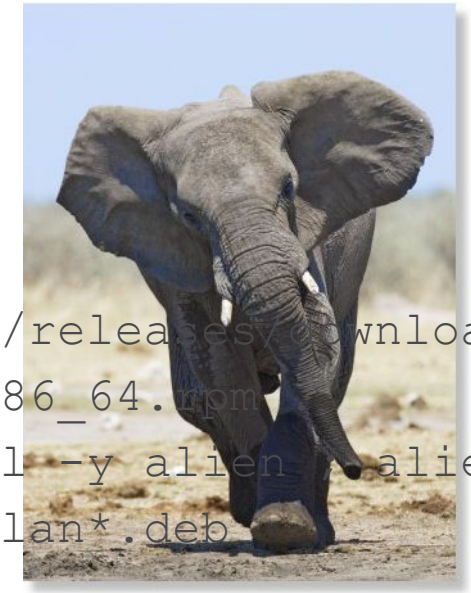
👉 You may know your data better, want stable plans...

Hints... how in PostgreSQL?

A harmless extension that has never been accepted in PG

Install pg_hint_plan (🙏 NTT OSS)

```
FROM docker.io/postgres:14
ADD
https://github.com/ossc-db/pg_hint_plan/releases/download/
/REL14_1_4_0/pg_hint_plan14-1.4-1.el8.x86_64.rpm
RUN  apt-get update -y ; apt-get install -y alien ; alien
./pg_hint_plan*.rpm ; dpkg -i pg-hint-plan*.deb
```



Postgres Optimizer

- PostgreSQL uses a cost-based optimizer, which utilizes data statistics, not static rules.
- The planner (optimizer) estimates costs of each possible execution plans for a SQL statement then the execution plan with the lowest cost is executed.
- The planner does its best to select the best execution plan, but is not always perfect, since it doesn't take into account some of the data properties or correlations between columns.

- pg_hint_plan makes it possible to tweak PostgreSQL execution plans using "hints" in SQL comments, as of /*+ SeqScan(a) */.

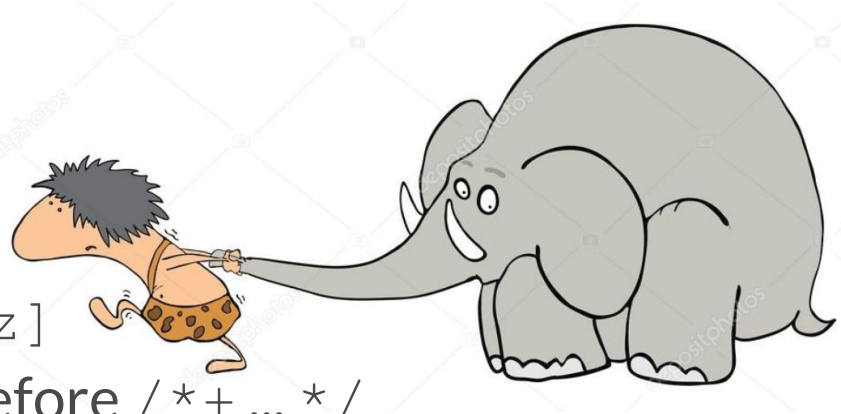
Hints as directives in SQL comments

Because SQL is declarative, hints are not SQL -> comments

```
/*+
Leading ( (...) ) NestLoop(...) IndexScan(...)
Set(...) Rows(...) Parallel(...)
*/
select ... ; insert... ; prepare... ; explain ...
```

Easy, if you understand that you rarely need a single hint

Where to put /*+ ... */



At the beginning of a command

```
[0-9 \t\n,_()A-Za-z]
```

are the only characters allowed before `/*+ ... */`

- Syntax errors stop parsing, no nested comment, no `--`
- In the PREPARE, not the EXECUTE
- ⚠️ with multi-statement commands `; ; ;` (in SQL) `\; \; \;` (in psql)

Hints reference tables and subqueries by their aliases

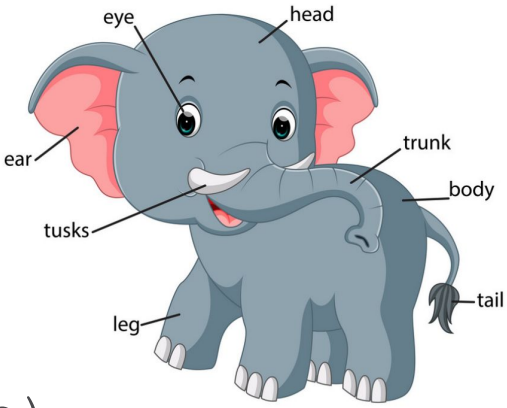
Many hints have a reference to tables

- by their alias (visible in execution plan)
- case sensitive (even with no quotes)
- Lists are not ordered:

```
HashJoin(a b c) = HashJoin(c a b)
```

- Nested Pairs are ordered:

```
Leading(  (a(b c)) ) != Leading(  ((a b)c)) )
```



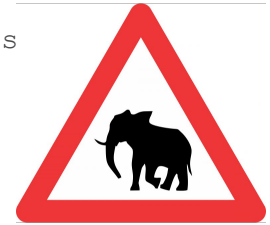
Hints reference indexes by their name (be careful if you rename them!)

A bad name ignore all indexes

```
postgres=# /*+ IndexScan (accounts accounts_email) */ explain select * from accounts
           where user_id=7;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on accounts  (cost=10000000000.00..10000000011.25 rows=1 width=520)

postgres=# /*+ IndexScan (accounts) */ explain select * from accounts where user_id=7;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using accounts_email_idx on accounts  (cost=0.14..8.16 rows=1 width=520)

postgres=# /*+ */ explain select * from accounts where user_id=7;
                                     QUERY PLAN
-------------------------------------------------------------------------------------------
 Index Only Scan using accounts_email_idx on accounts  (cost=0.14..8.16 rows=1 width=520)
```



Troubleshooting: errors and log

By default:

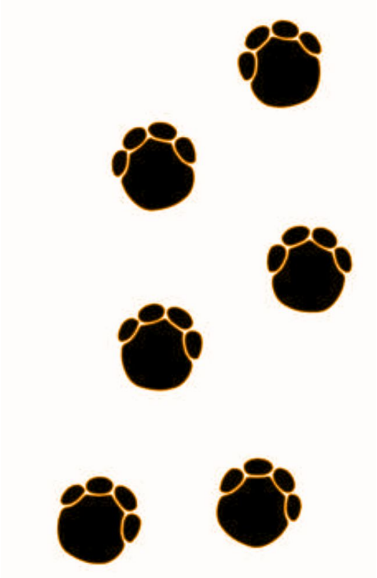
```
INFO:  pg_hint_plan: hint syntax error
```

More info in the log (`on` or `verbose`):

```
set pg_hint_plan.debug_print=verbose;
```

To the client (`pg_hint_plan.message_level` defaults to log):

```
set client_min_messages = log;
```



What does a hint do?

Hints do not force anything

It can sets high cost for the unwanted access paths

Is evaluated during the query planning process

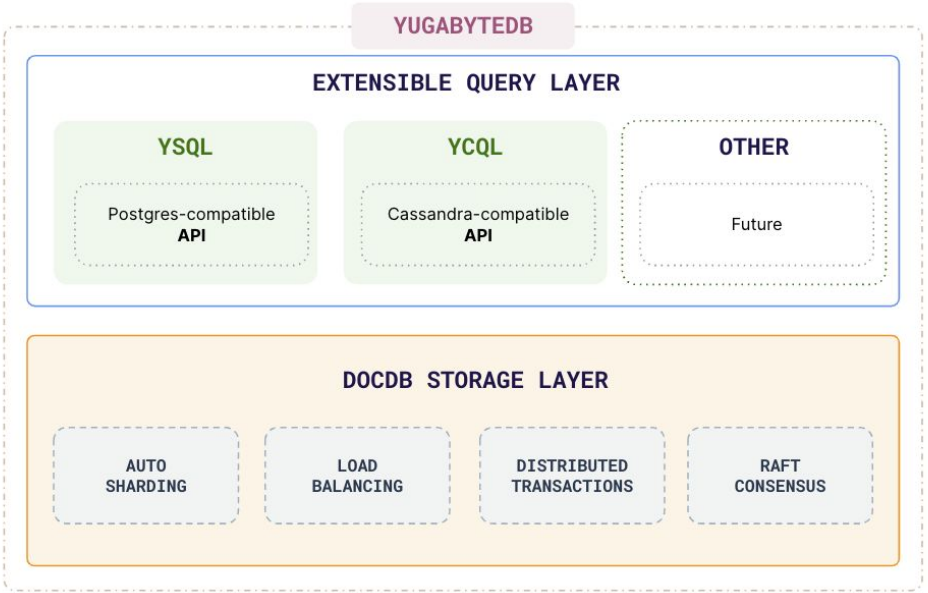
https://github.com/postgres/postgres/blob/master/src/backend/optimizer/path/costsize.c

```
131   Cost        disable_cost = 1.0e10;
```

Demo:

- Scan Method (Specify Indexes, Index list)
- Join Methods
- Join Order
- Setting Work Mem
- Setting GUC Planner

YugabyteDB



PostgreSQL Compatibility

Resilience and High Availability

Horizontal Scalability

Geographic Distribution

ACID Transactions

Security

Postgres query planner

- The task of the *planner/optimizer* is to create an optimal execution plan
- Sample table

```
yugabyte=# \d t1
                Table "public.t1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t1_pkey" PRIMARY KEY, lsm (id HASH)
    "t1_val" lsm (val HASH)
    "t1_val_2" lsm (val HASH) WHERE val = 2
```

- Simple index lookup query

```
yugabyte=# EXPLAIN SELECT val FROM t1 WHERE val=2;
                            QUERY PLAN
-------------------------------------------------------------------
 Index Only Scan using t1_val on t1  (cost=0.00..5.22 rows=10 width=8)
   Index Cond: (val = 2)
(2 rows)
```

Alternate query plan

Cost-based optimizer

pg_hint_plan PostgreSQL extension

- pg_hint_plan extension allows users to provide hints to the planner in order to control the execution plan of a query
- Hinting phrases of a special form present in SQL statements
- Begins with the character sequence "/*+" and ends with "*/"
- Sample table

```
yugabyte=# \d t1
                Table "public.t1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t1_pkey" PRIMARY KEY, lsm (id HASH)
    "t1_val" lsm (val HASH)
    "t1_val_2" lsm (val HASH) WHERE val = 2
```

Hint Phrase

- Index lookup query

```
yugabyte=# /*+ IndexScan(t1 t1_val_2)*/ EXPLAIN SELECT val FROM t1 WHERE val=2;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using t1_val_2 on t1  (cost=0.00..102.00 rows=1000 width=8)
(1 row)
```

Preferred Index

pg_hint_plan features

Features of `pg_hint_plan`

pg_hint_plan's features can be leveraged to control

1. Scan Methods (Index selection)
2. Join Methods (Nested Loop/Hash/Merge…)
3. Joining Order

In a similar fashion, users can leverage pg_hint_plan

4. To configure postgres working memory
5. To set GUC planner method config knobs at statement level

pg_hint_plan has features to help queries pick hints without adding comments

6. Hint tables

Scan methods

Enforce the scanning method on tables when specified along with appropriate hint phrases.

Option	Value Notes
SeqScan(table)	Enable SeqScan on the table.
NoSeqScan(table)	Do not enable SeqScan on the table.
IndexScan(table)	Enable IndexScan on the table.
IndexScan(table idx)	Enable IndexScan on the table using the index idx.
NoIndexScan(table)	Do not enable IndexScan on the table.
IndexOnlyScan(table)	Enable IndexOnlyScan on the table.
NoIndexOnlyScan(table)	Do not enable IndexOnlyScan on the table.
IndexScanRegexp(table regex)	Enable index scan on the table whose indices match with the regular expression defined by regex.
IndexOnlyScanRegexp(table regex)	Do not enable index scan on the table whose indices match with the regular expression defined by regex.

Scan methods (example)

```
                Table "public.t1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t1_pkey" PRIMARY KEY, lsm (id HASH)
       "t1_val" lsm (val HASH)
```

10,000 ROWS

```
                Table "public.t2"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t2_pkey" PRIMARY KEY, lsm (id HASH)
       "t2_val" lsm (val HASH)
```

1000 ROWS

Scan methods (example)

Example 1

```
yugabyte=# /*+SeqScan(t2)*/
EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE t1.id =
t2.id;
              QUERY PLAN
-------------------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(4 rows)
```

Example 2

```
yugabyte=# /*+SeqScan(t1)IndexScan(t2)*/
yugabyte-# EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE
t1.id = t2.id;
              QUERY PLAN
-------------------------------------
 Nested Loop
   ->  Seq Scan on t1
   ->  Index Scan using t2_pkey on t2
         Index Cond: (id = t1.id)
(4 rows)
```

Example 3

```
yugabyte=# /*+NoIndexScan(t1)*/
yugabyte-# EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE t1.id =
t2.id;
              QUERY PLAN
-------------------------------------
 Nested Loop
   ->  Seq Scan on t1
   ->  Index Scan using t2_pkey on t2
         Index Cond: (id = t1.id)
(4 rows)
```

Scan methods (Specifying Indexes example)

```
                Table "public.t3"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t3_pkey" PRIMARY KEY, lsm (id HASH)
    "t3_id1" lsm (id HASH)
    "t3_id2" lsm (id HASH)
    "t3_id3" lsm (id HASH)
       "t3_val" lsm (val HASH)

                100 ROWS
```

Scan methods (Specifying Indexes example)

Baseline Query

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t3
WHERE t3.id = 1;
           QUERY PLAN
--------------------------------
 Index Scan using t3_pkey on t3
   Index Cond: (id = 1)
(2 rows)
```

Scan methods (Specifying Indexes example)

Baseline Query

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t3
WHERE t3.id = 1;
           QUERY PLAN
--------------------------------
 Index Scan using t3_pkey on t3
   Index Cond: (id = 1)
(2 rows)
```

t3_id2 Index

```
yugabyte=# /*+IndexScan(t3 t3_id2)*/
yugabyte-# EXPLAIN (COSTS false) SELECT * FROM t3
WHERE t3.id = 1;
           QUERY PLAN
--------------------------------
 Index Scan using t3_id2 on t3
   Index Cond: (id = 1)
(2 rows)
```

Scan methods (Specifying Indexes example)

Baseline Query

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t3
WHERE t3.id = 1;
           QUERY PLAN
--------------------------------
 Index Scan using t3_pkey on t3
   Index Cond: (id = 1)
(2 rows)
```

no_exist Index

```
yugabyte=# /*+IndexScan(t3 no_exist)*/
yugabyte-# EXPLAIN (COSTS false) SELECT * FROM t3
WHERE t3.id = 1;
     QUERY PLAN
--------------------
 Seq Scan on t3
   Filter: (id = 1)
(2 rows)
```

Scan methods (Index list example)

```
yugabyte=# \d t3
                Table "public.t3"
 Column |  Type   | Collation | Nullable |
Default
--------+---------+-----------+----------+---
------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t3_pkey" PRIMARY KEY, lsm (id HASH)
    "t3_val" lsm (val HASH)
    "t3_val_2" lsm (val HASH) WHERE val = 2
    "t3_val_3" lsm (val HASH) WHERE val = 3
    "t3_val_4" lsm (val HASH) WHERE val = 4
```

Scan methods (Index list example)

```
yugabyte=# EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 1;
            QUERY PLAN
-----------------------------------
 Index Only Scan using t3_val on t3
   Index Cond: (val = 1)
(2 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 2;
            QUERY PLAN
-----------------------------------
 Index Only Scan using t3_val on t3
   Index Cond: (val = 1)
(2 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 3;
            QUERY PLAN
-----------------------------------
 Index Only Scan using t3_val on t3
   Index Cond: (val = 3)
(2 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 4;
            QUERY PLAN
-----------------------------------
 Index Only Scan using t3_val on t3
   Index Cond: (val = 4)
(2 rows)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT * FROM t3 WHERE t3.val = 1;
     QUERY PLAN
--------------------
 Seq Scan on t3
   Filter: (val = 1)
(2 rows)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT * FROM t3 WHERE t3.val = 2;
           QUERY PLAN
--------------------------------
 Index Scan using t3_val_2 on t3
(1 row)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 3;
           QUERY PLAN
--------------------------------
 Index Scan using t3_val_3 on t3
(1 row)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT val FROM t3 WHERE t3.val = 4;
           QUERY PLAN
--------------------------------
 Index Scan using t3_val_4 on t3
(1 row)
```

Scan methods (Index list example)

```
yugabyte=# EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 2;
            QUERY PLAN
-----------------------------------
 Aggregate
   ->  Index Only Scan using t3_val on t3
         Index Cond: (val = 2)
(3 rows)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 2;
              QUERY PLAN
---------------------------------------
 Aggregate
   ->  Index Scan using t3_val_2 on t3
(2 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 3;
            QUERY PLAN
-----------------------------------
 Aggregate
   ->  Index Only Scan using t3_val on t3
         Index Cond: (val = 3)
(3 rows)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 3;
              QUERY PLAN
---------------------------------------
 Aggregate
   ->  Index Scan using t3_val_3 on t3
(2 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 4;
            QUERY PLAN
-----------------------------------
 Aggregate
   ->  Index Only Scan using t3_val on t3
         Index Cond: (val = 4)
(3 rows)
```

```
yugabyte=# /*+IndexScan(t3 t3_val_2 t3_val_3 t3_val_4)*/
EXPLAIN (COSTS false) SELECT count(*) FROM t3 WHERE t3.val = 4;
              QUERY PLAN
---------------------------------------
 Aggregate
   ->  Index Scan using t3_val_4 on t3
(2 rows)
```

Join methods

Enforces the join methods for SQL statements.

Option	Value Notes
HashJoin(t1 t2 t3 ...)	Join t1, t2, and t3 using HashJoin.
NoHashJoin(t1 t2 t3 ...)	Do not join t1, t2, and t3 using HashJoin.
NestLoop(t1 t2 t3 ...)	Join t1, t2, and t3 using NestLoop join.
NoNestLoop(t1 t2 t3 ...)	Do not join t1, t2, and t3 using NestLoop join.

Join methods (example)

Table "public.t1"

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t1_pkey" PRIMARY KEY, lsm (id HASH)
```

10,000 ROWS

Table "public.t2"

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t2_pkey" PRIMARY KEY, lsm (id HASH)
```

1000 ROWS

Join methods (example)

Default Execution

```
yugabyte=# EXPLAIN ANALYZE SELECT * FROM t1, t2 WHERE t1.id = t2.id;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..210.40 rows=1000 width=16) (actual time=5.376..200.887 rows=1000 loops=1)
   ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.902..5.477 rows=1000 loops=1)
   ->  Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (actual time=0.188..0.188 rows=1 loops=1000)
         Index Cond: (id = t2.id)
 Planning Time: 0.097 ms
 Execution Time: 201.082 ms
(6 rows)
```

Optimized Execution

```
yugabyte=# /*+HashJoin(t1 t2)*/ EXPLAIN ANALYZE SELECT * FROM t1, t2 WHERE t1.id = t2.id;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=112.50..1138.76 rows=1000 width=16) (actual time=19.865..80.962 rows=1000 loops=1)
   Hash Cond: (t1.id = t2.id)
   ->  Seq Scan on t1  (cost=0.00..1000.00 rows=10000 width=8) (actual time=9.579..69.479 rows=10000 loops=1)
   ->  Hash  (cost=100.00..100.00 rows=1000 width=8) (actual time=9.107..9.107 rows=1000 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 48kB
         ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.516..8.976 rows=1000 loops=1)
 Planning Time: 1.265 ms
 Execution Time: 81.391 ms
(8 rows)
```

Decreased Execution Time

Scan + Join methods (example)

Example 1

```
yugabyte=# /*+NestLoop(t2 t3 t1) SeqScan(t3) SeqScan(t2)*/ EXPLAIN (COSTS false) SELECT *
FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
              QUERY PLAN
-------------------------------------
 Nested Loop
   ->  Hash Join
         Hash Cond: (t2.id = t3.id)
         ->  Seq Scan on t2
         ->  Hash
               ->  Seq Scan on t3
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(8 rows)
```

Join order (example)

Table "public.t1"

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t1_pkey" PRIMARY KEY, lsm (id HASH)
```

10,000 ROWS

Table "public.t2"

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t2_pkey" PRIMARY KEY, lsm (id HASH)
```

1000 ROWS

Table "public.t3"

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 val    | integer |           |          |
Indexes:
    "t3_pkey" PRIMARY KEY, lsm (id HASH)
```

100 ROWS

Join order (example)

Default Execution

```
yugabyte=# EXPLAIN ANALYZE SELECT * FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
                                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..320.80 rows=100 width=24) (actual time=362.519..362.519 rows=0 loops=1)
   ->  Nested Loop  (cost=0.00..210.40 rows=1000 width=16) (actual time=5.317..191.418 rows=1000 loops=1)
         ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.865..5.853 rows=1000 loops=1)
         ->  Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (actual time=0.178..0.178 rows=1 loops=1000)
               Index Cond: (id = t2.id)
   ->  Index Scan using t3_pkey on t3  (cost=0.00..0.11 rows=1 width=8) (actual time=0.164..0.164 rows=0 loops=1000)
         Index Cond: (id = t1.id)
 Planning Time: 0.169 ms
 Execution Time: 362.561 ms
(9 rows)
```

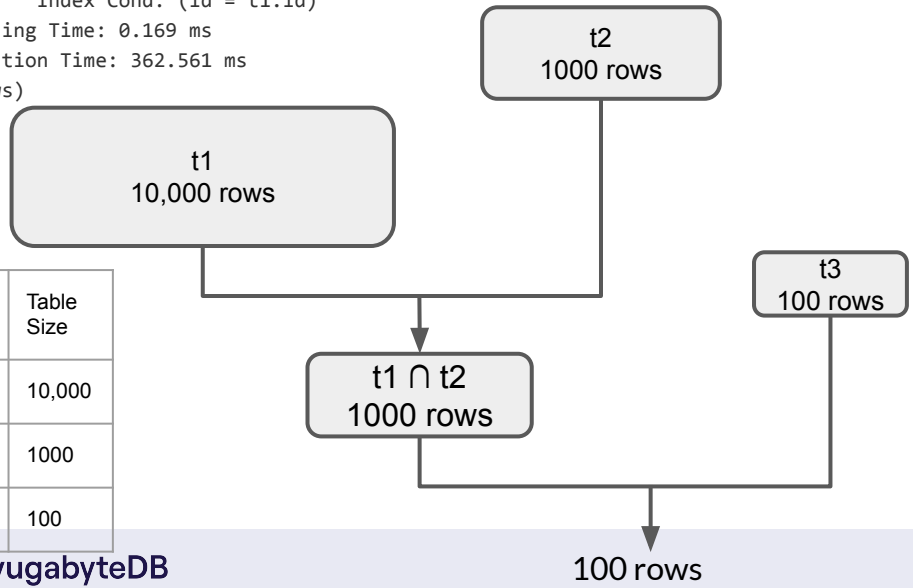


Table Name	Table Size
t1	10,000
t2	1000
t3	100

Join order (example)

Default Execution

```
yugabyte=# EXPLAIN ANALYZE SELECT * FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..320.80 rows=100 width=24) (actual time=362.519..362.519 rows=0 loops=1)
   ->  Nested Loop  (cost=0.00..210.40 rows=1000 width=16) (actual time=5.317..191.418 rows=1000 loops=1)
         -> Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.865..5.853 rows=1000 loops=1)
         -> Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (actual time=0.178..0.178 rows=1 loops=1000)
               Index Cond: (id = t2.id)
   ->  Index Scan using t3_pkey on t3  (cost=0.00..0.11 rows=1 width=8) (actual time=0.164..0.164 rows=0 loops=1000)
         Index Cond: (id = t1.id)
 Planning Time: 0.169 ms
 Execution Time: 362.561 ms
(9 rows)
```

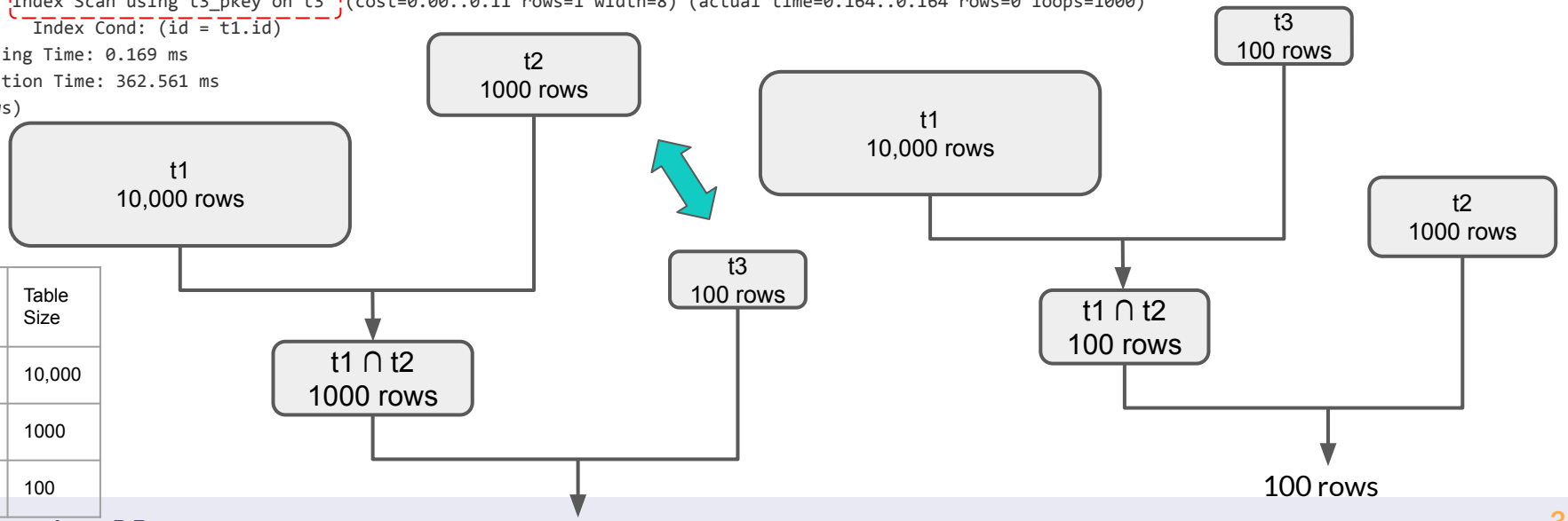


Table Name	Table Size
t1	10,000
t2	1000
t3	100

Join order (example)

Default Execution

```
yugabyte=# EXPLAIN ANALYZE SELECT * FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..320.80 rows=100 width=24) (actual time=362.519..362.519 rows=0 loops=1)
   ->  Nested Loop  (cost=0.00..210.40 rows=1000 width=16) (actual time=5.317..191.418 rows=1000 loops=1)
         -> Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.865..5.853 rows=1000 loops=1)
         -> Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (actual time=0.178..0.178 rows=1 loops=1000)
               Index Cond: (id = t2.id)
   ->  Index Scan using t3_pkey on t3  (cost=0.00..0.11 rows=1 width=8) (actual time=0.164..0.164 rows=0 loops=1000)
         Index Cond: (id = t1.id)
 Planning Time: 0.169 ms
 Execution Time: 362.561 ms
(9 rows)
```

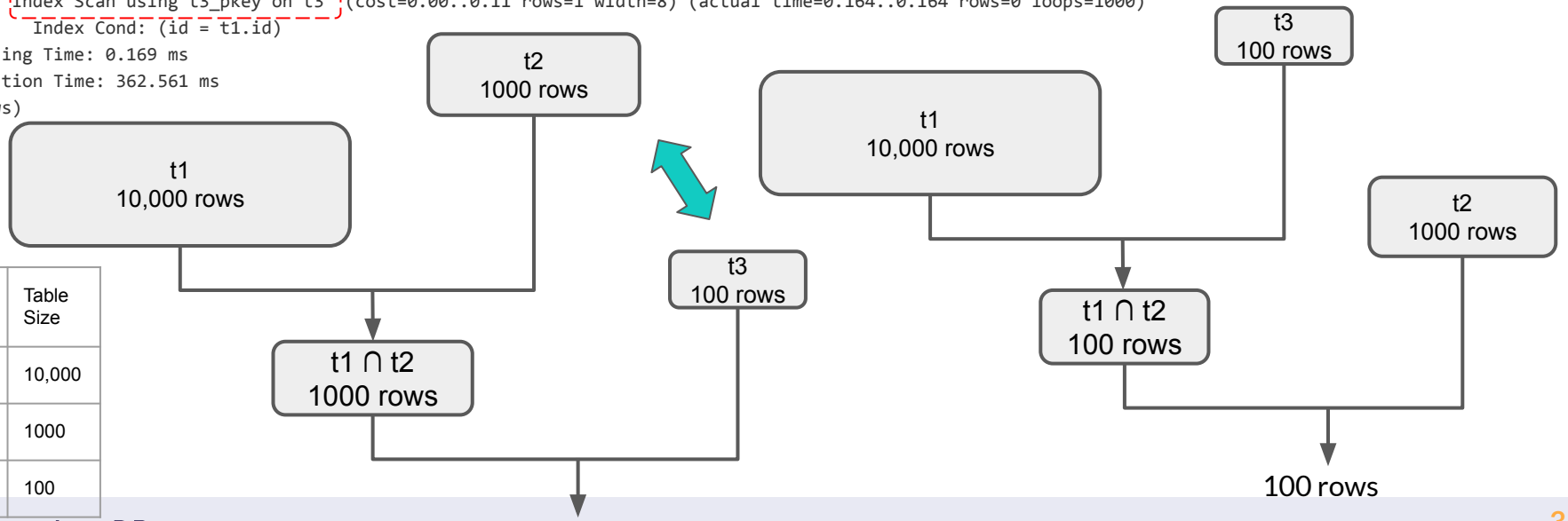


Table Name	Table Size
t1	10,000
t2	1000
t3	100

Join order (example)

Default Execution

```
yugabyte=# EXPLAIN ANALYZE SELECT * FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..320.80 rows=100 width=24) (actual time=362.519..362.519 rows=0 loops=1)
   ->  Nested Loop  (cost=0.00..210.40 rows=1000 width=16) (actual time=5.317..191.418 rows=1000 loops=1)
         ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=4.865..5.853 rows=1000 loops=1)
         ->  Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (actual time=0.178..0.178 rows=1 loops=1000)
               Index Cond: (id = t2.id)
   ->  Index Scan using t3_pkey on t3  (cost=0.00..0.11 rows=1 width=8) (actual time=0.164..0.164 rows=0 loops=1000)
         Index Cond: (id = t1.id)
 Planning Time: 0.169 ms
 Execution Time: 362.561 ms
(9 rows)
```

Optimized Execution

```
yugabyte=# /*+Leading(t3 t2 t1)*/ EXPLAIN ANALYZE SELECT * FROM t1, t2, t3 WHERE t1.id = t2.id AND t1.id = t3.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..324.29 rows=100 width=24) (actual time=183.377..183.377 rows=0 loops=1)
   ->  Nested Loop  (cost=0.00..213.89 rows=1000 width=16) (actual time=183.377..183.377 rows=0 loops=1)
         ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=8) (actual time=5.354..6.271 rows=1000 loops=1)
         ->  Index Scan using t3_pkey on t3  (cost=0.00..0.11 rows=1 width=8) (actual time=0.170..0.170 rows=0 loops=1000)
               Index Cond: (id = t2.id)
   ->  Index Scan using t1_pkey on t1  (cost=0.00..1.10 rows=10 width=8) (never executed)
         Index Cond: (id = t2.id)
 Planning Time: 0.225 ms
 Execution Time: 183.420 ms
(9 rows)
```

Decreased Execution Time

Join order (example)

Example 1

```
yugabyte=# /*+Leading(t1 t2 t3)*/
EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3
WHERE t1.id = t2.id AND t1.id = t3.id;
                 QUERY PLAN
--------------------------------------------
 Nested Loop
   ->  Nested Loop
         ->  Seq Scan on t1
         ->  Index Scan using t2_pkey on t2
               Index Cond: (id = t1.id)
   ->  Index Scan using t3_pkey on t3
         Index Cond: (id = t1.id)
(7 rows)
```

Example 2

```
yugabyte=# /*+Leading(t2 t3 t1)*/
EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3 WHERE
t1.id = t2.id AND t1.id = t3.id;
                 QUERY PLAN
--------------------------------------------
 Nested Loop
   ->  Nested Loop
         ->  Seq Scan on t2
         ->  Index Scan using t3_pkey on t3
               Index Cond: (id = t2.id)
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(7 rows)
```

Setting work mem (example)

Example 1

```
yugabyte=# /*+Set(work_mem "1MB")*/
EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE t1.id = t2.id;
              QUERY PLAN
-------------------------------------
 Nested Loop
   ->  Seq Scan on t1
   ->  Index Scan using t2_pkey on t2
         Index Cond: (id = t1.id)
(4 rows)
```

Setting GUC planner method config (example)

Example 1

```
yugabyte=# /*+Set(enable_indexscan off)*/
EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE t1.id = t2.id;
          QUERY PLAN
------------------------------
 Hash Join
   Hash Cond: (t1.id = t2.id)
   ->  Seq Scan on t1
   ->  Hash
         ->  Seq Scan on t2
(5 rows)
```

Setting GUC planner method config (example)

Option	Value Notes
enable_hashagg	
enable_hashjoin	
enable_indexscan	
enable_indexonlyscan	
enable_material	
enable_nestloop	
enable_partition_pruning	
enable_partitionwise_join	
enable_partitionwise_aggregate	
enable_seqscan	
enable_sort	

Hint Tables

- Adding comments to every query becomes cumbersome
- Group similar type of queries and enable pg_hint_plan
- Grouping information is stored in a table called hint_plan.hints

Hint Tables (example)

- Adding comments to every query becomes cumbersome
- Group similar type of queries and enable `pg_hint_plan`
- Grouping information is stored in a table called `hint_plan.hints`

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE t1.id =
t2.id;
                QUERY PLAN
-----------------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(4 rows)
```

Hint Tables (example)

- Adding comments to every query becomes cumbersome
- Group similar type of queries and enable pg_hint_plan
- Grouping information is stored in a table called hint_plan.hints

INSERT INTO hint_plan.hints

```
yugabyte=# SELECT * FROM hint_plan.hints;
-[ RECORD 1 ]-----+-------------------------------------------------
id                | 108
norm_query_string | EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE
t1.id = t2.id;
application_name  |
hints             | HashJoin(t1 t2)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3 WHERE t1.id
= t2.id AND t1.id = t3.id;
          QUERY PLAN
-------------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(4 rows)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3 WHERE t1.id
= t2.id AND t1.id = t3.id;
          QUERY PLAN
-------------------------------
 Hash Join
   Hash Cond: (t1.id = t2.id)
   ->  Seq Scan on t1
   ->  Hash
         ->  Seq Scan on t2
(5 rows)
```

Change in query plan without hints

Hint Tables (example)

- Adding comments to every query becomes cumbersome
- Group similar type of queries and enable
- Grouping information is stored in a table called

INSERT INTO hint_plan.hints

```
yugabyte=# SELECT * FROM hint_plan.hints;
-[ RECORD 1 ]-----+-------------------------------------------------
id                | 108
norm_query_string | EXPLAIN (COSTS false) SELECT * FROM t1,
t2 WHERE t1.id = t2.id;
application_name  |
hints             | HashJoin(t1 t2)
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3 WHERE t1.id
= t2.id AND t1.id = t3.id;
```

pg_hint_plan

```
        QUERY PLAN
-----------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using t1_pkey on t1
         Index Cond: (id = t2.id)
(4 rows)
```

hint_plan.hints

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3
WHERE t1.id = t2.id AND t1.id = t3.id;
```

```
        QUERY PLAN
-----------------------------
 Hash Join
   Hash Cond: (t1.id = t2.id)
   ->  Seq Scan on t1
   ->  Hash
         ->  Seq Scan on t2
(5 rows)
```

Normalized query strings should match verbatim

e in query plan
thout hints

Hint Tables (example)

- Adding comments to every query becomes cumbersome
- Group similar type of queries and enable pg_hint_plan
- Grouping information is stored in a table called hint_plan.hints

```
norm_query_string | EXPLAIN (COSTS false) SELECT * FROM t1, t2 WHERE
t1.id = t2.id;
```

```
yugabyte=# EXPLAIN (COSTS false) SELECT * FROM t1, t2, t3 WHERE t1.id
= t2.id AND t1.id = t3.id;
```

Normalized query strings match verbatim

Hint Tables (example)

Example 1

```
yugabyte=# INSERT INTO hint_plan.hints
      (norm_query_string,
       application_name,
       hints)
      VALUES
      ('SELECT * FROM t1 WHERE t1.id = ?;',
       '',
       'SeqScan(t1)');
INSERT 0 1

yugabyte=# INSERT INTO hint_plan.hints
      (norm_query_string,
       application_name,
       hints)
      VALUES
      ('EXPLAIN (COSTS false) SELECT id FROM t1 WHERE t1.id =
?;',
       '',
       'IndexScan(t1)');
INSERT 0 1
```

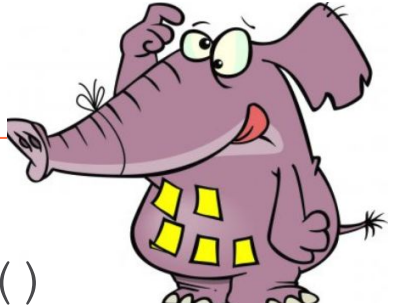
```
yugabyte=# select * from hint_plan.hints;

-[ RECORD 1 ]
-----+-------------------------------------------
id               | 1
norm_query_string | EXPLAIN (COSTS false) SELECT *
FROM t1 WHERE t1.id = ?;
application_name  |
hints             | SeqScan(t1)

-[ RECORD 2 ]
-----+-------------------------------------------
id               | 2
norm_query_string | EXPLAIN (COSTS false) SELECT id
FROM t1 WHERE t1.id = ?;
application_name  |
hints             | IndexScan(t1)
```

an hint on full hinting

For *n* table **aliases** in your (sub-)query



- *n*-1 nested pair of `(outer inner)` in `Leading()`
- for each pair: `NestLoop()`, `HashJoin()` or `MergeJoin()` they will have from 2 to *n* aliases (order doesn't matter)
- *n* scan method SeqScan(), IndexScan(), IndexOnlyScan() IndexScanRegexp(), ...

  https://github.com/ossc-db/pg_hint_plan#hints-list

🧮 count 6*n*-2 closing (or opening) parentheses

Set parameters at query level

Example:

You know that Partition-wise join is good for one query but don't want to take more CPU and memory for other queries

```
/*+
 Set(enable_partitionwise_join true)
*/
```

no risk to forget to reset it back after

⚠️for planning only, not execution

What if you cannot change the query?

```
create extension pg_hint_plan;

insert into hint_plan.hints
 (norm_query_string, application_name, hints) values (
 $sql$select * from table where a=$1 and b=?$sql$,
 'my_app', 'Leading( (a b) )'
);

set pg_hint_plan.enable_hint_table=on;
```

Applies hints to your application query

by matching the command text

- `$1`,`$2` are for prepared statements parameters
- `?` is for literals replaced before matching a query
- No final `;` except if there's one in your command

Hints in view?

Hints are ignored in views but applied in functions

- explain the view to get the aliases
- create a function on top of the view, with the hints

or create the function with the view text and a view on top of it

```
create or replace function myview()
returns setof myview as
$$
   /*+ NestLoop(demo1 demo2) */
   select * from myview;
$$ language sql;
```

Partitions?

```
Append
  ->  Index Scan using i1 on p1  p
        Index Cond: (c = 1)
  ->  Index Scan using i2 on p2  p_1
        Index Cond: (c = 1)
```

Table hinted with the global table alias: `/*+ IndexScan( p ) */`

Index hinted with the index partition name: `/*+ IndexScan( p  i1) */`

For multiple partitions, use a regexp `/*+ IndexScanRegexp( p  i?) */`

Core message:

You may need hints, one day, maybe in emergency

(short-term workaround with no side effect on other queries)

👉better have it installed and know how it works

Great tool to experiment and learn about the query planner

amchauhan@yugabyte.com

E-mail:
amchauhan@yugabyte.com

Blogs:
blog.yugabyte.com

LinkedIn:
www.linkedin.com/in/thechauhan

Community Slack / Github:
www.yugabyte.com/community

Amit Chauhan, Principal Solution Engineer

yugabyteDB