Index 01

“Ego kills knowledge,
as knowledge requires learning,
and learning requires humility”
- Rolsey

RONIN™
ENGINEER

Outline

1. Query Optimization
2. Query Execution
   - SQL Execution
   - Execution Plan
3. Practices

1. Query Optimization

1.1. Query Optimization

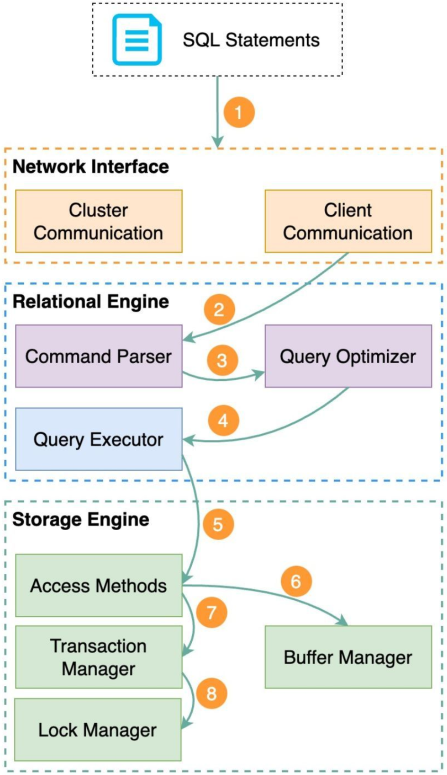
- There are two parts:
  - **Direct Query Optimization: changes to queries and indexes**
    - Rewrite query
    - Index
    - …
  - **Indirect Query Optimization: changes to data and access patterns**
    - Changes to data: reducing the size of data, move old data to cold storage
    - Denormalization
    - Partitioning
    - Defragment (operation)
    - …
- Optimize in the order: direct → indirect

2. Query Execution

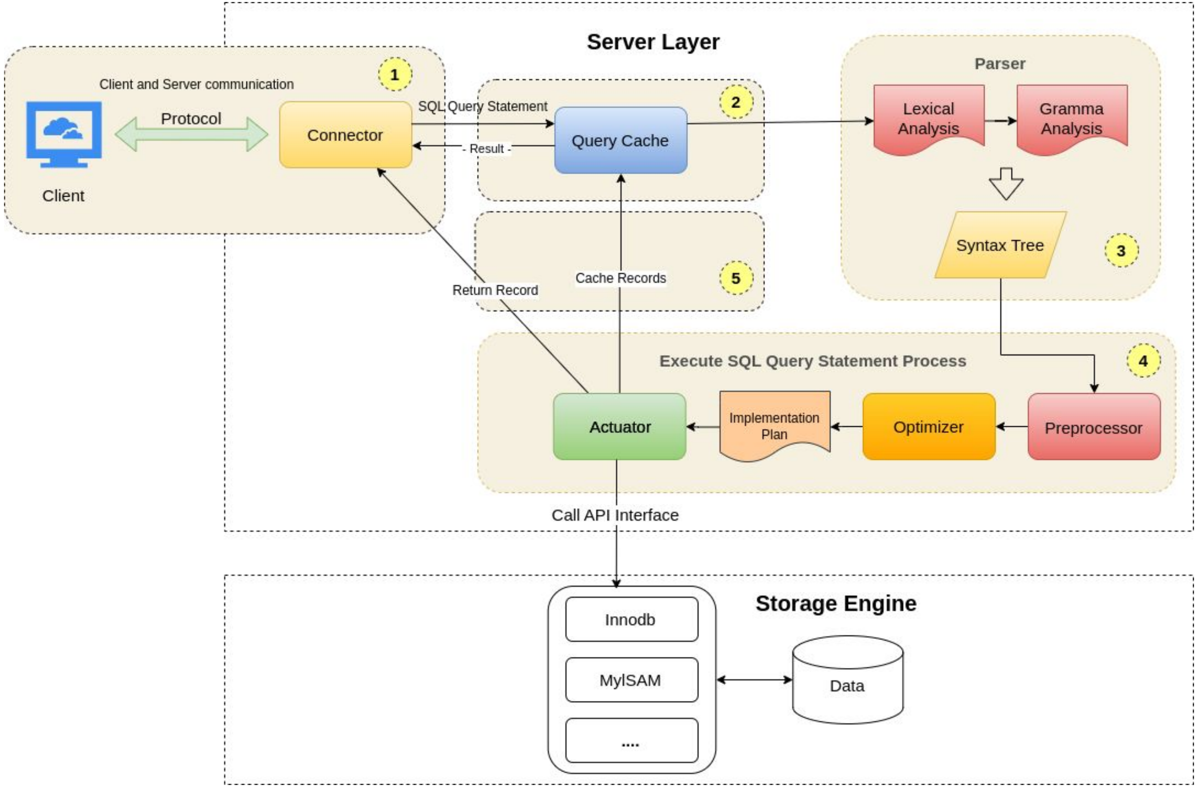
2.1. SQL Execution

2.1.1. How SQL Executed in DB?

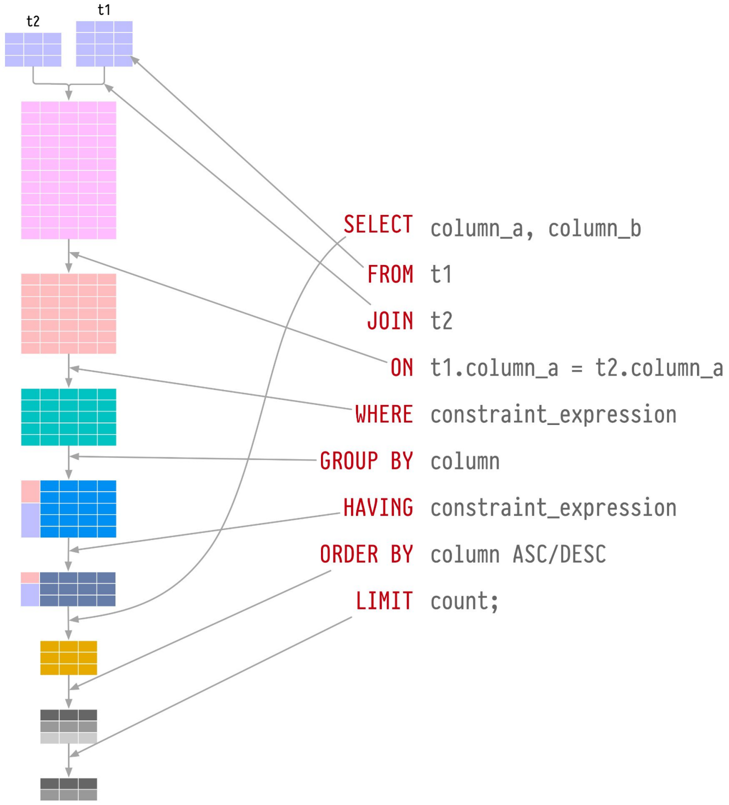
- Step 1: Client submit statement to RDBMS
- Step 2: RDBMS received SQL from Network Interface
- Step 3: The Parser preprocesses, parses SQL into a query tree
- Step 4: **Optimizer generates execution plans then choose one plan**
- Step 5: Executor retrieves data from storage engine based on the plan
- Step 6: If statement is read-only → Buffer Manager
- Step 7: If statement is a write → Transaction Manager
- Step 8: During a transaction, Lock manager ensure ACID properties



2.1.1. How SQL Executed in DB?



2.1.2. SQL Execution Order

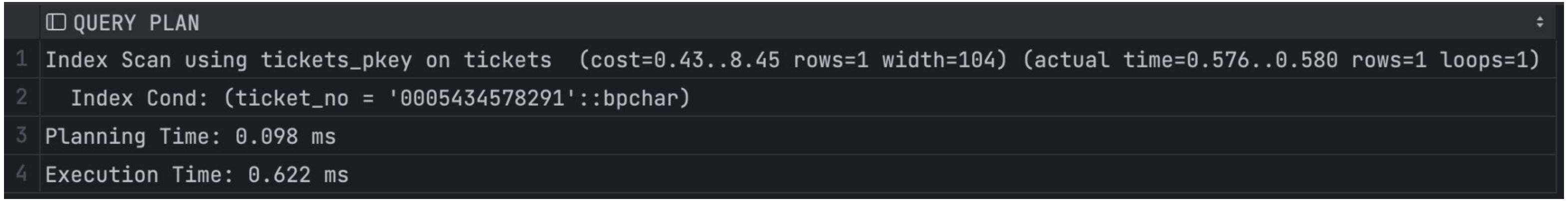


```
SELECT column_a, column_b
  FROM t1
  JOIN t2
    ON t1.column_a = t2.column_a
 WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
 LIMIT count;
```

2.2. Execution Plan

2.2.1. Execution Plan (Postgres)

- Execution plan is a detailed, step-by-step description of how RDBMS will execute a specific SQL query → **important to optimize query**
- Syntax:
  - EXPLAIN: to get basic information about the plan
  - ANALYZE: to get more concrete information about the plan
  - BUFFERS: information about cache hits/misses
  - FORMAT: (recommended) reformat output to YAML / JSON
  - Example: EXPLAIN (ANALYZE, BUFFERS, FORMAT YAML) SELECT …



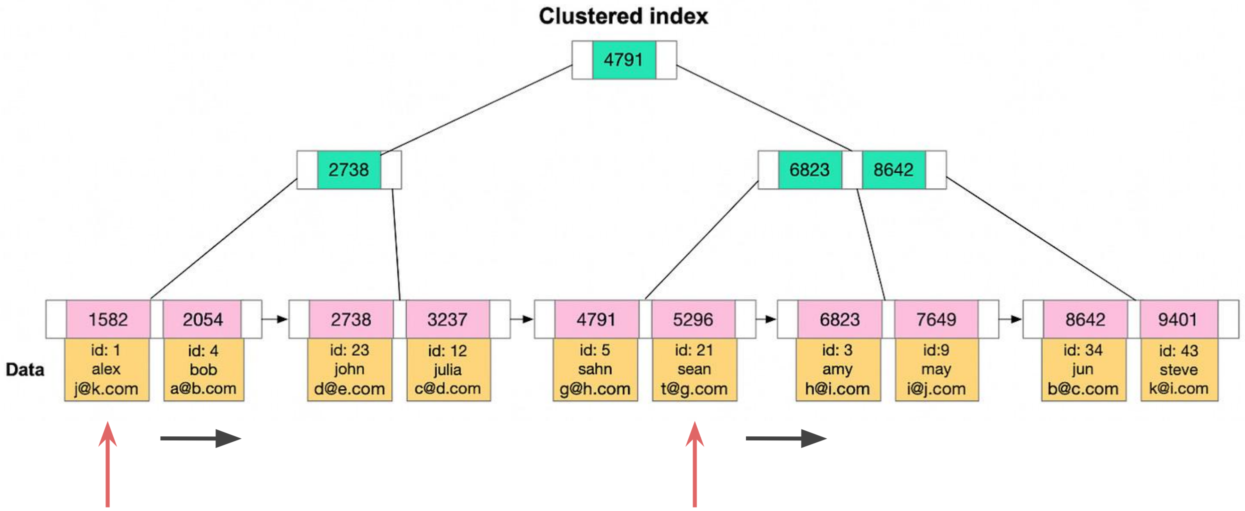
```
QUERY PLAN
Index Scan using tickets_pkey on tickets  (cost=0.43..8.45 rows=1 width=104) (actual time=0.576..0.580 rows=1 loops=1)
  Index Cond: (ticket_no = '0005434578291'::bpchar)
Planning Time: 0.098 ms
Execution Time: 0.622 ms
```

2.2.2. Types of Scanning

- Sequential Scan
- Index Scan
- Index Only Scan
- Bitmap Index Scan + Bitmap Heap Scan

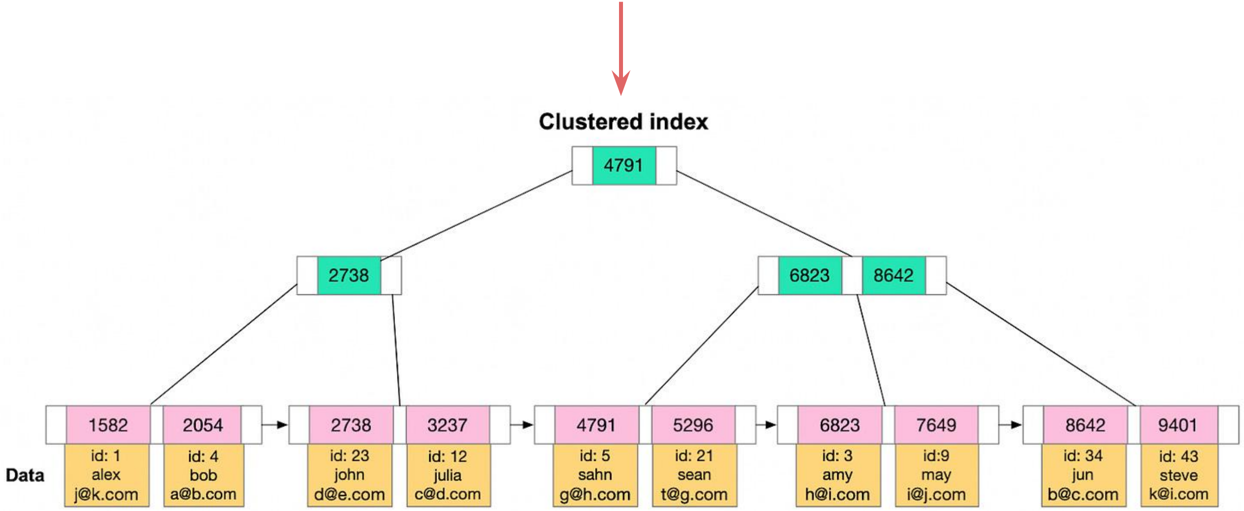
2.2.2. Types of Scanning

- **Sequential Scan**: scanning through the source table row-by-row without using any index
- **Parallel Sequential Scan:** using multi workers (threads) to scan



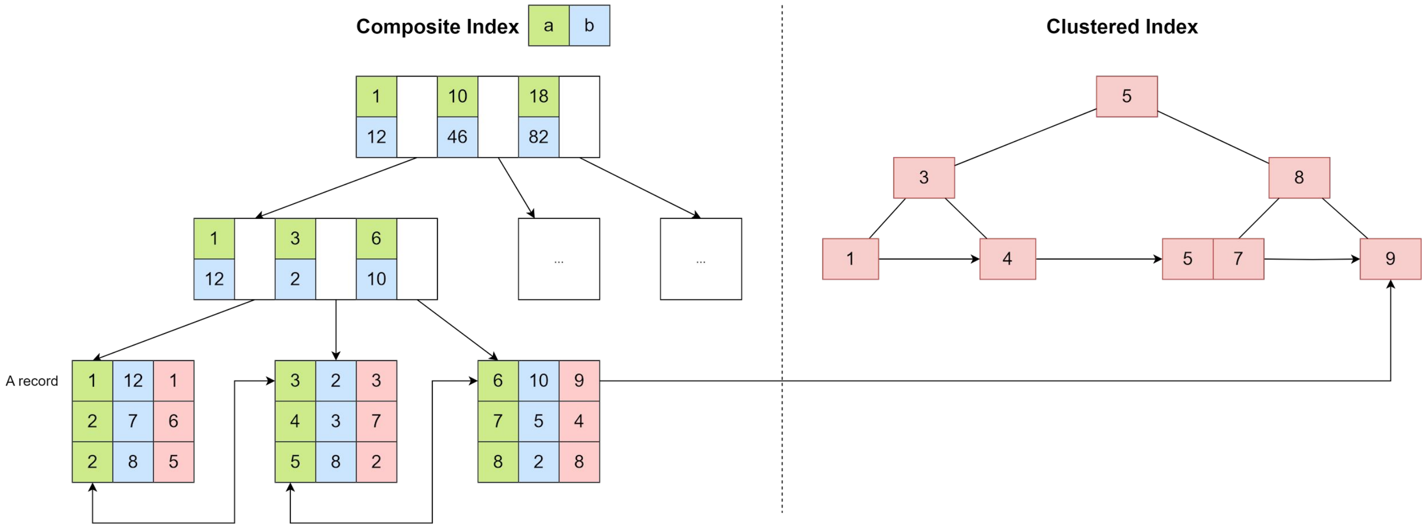
2.2.2. Types of Scanning

- **Index Scan**: using an index to determines which table rows match, and then retrieves the rows from the table



2.2.2. Types of Scanning

- **Index Only Scan**: retrieving the actual query result directly from the index, and avoids accessing the table itself for the requested data

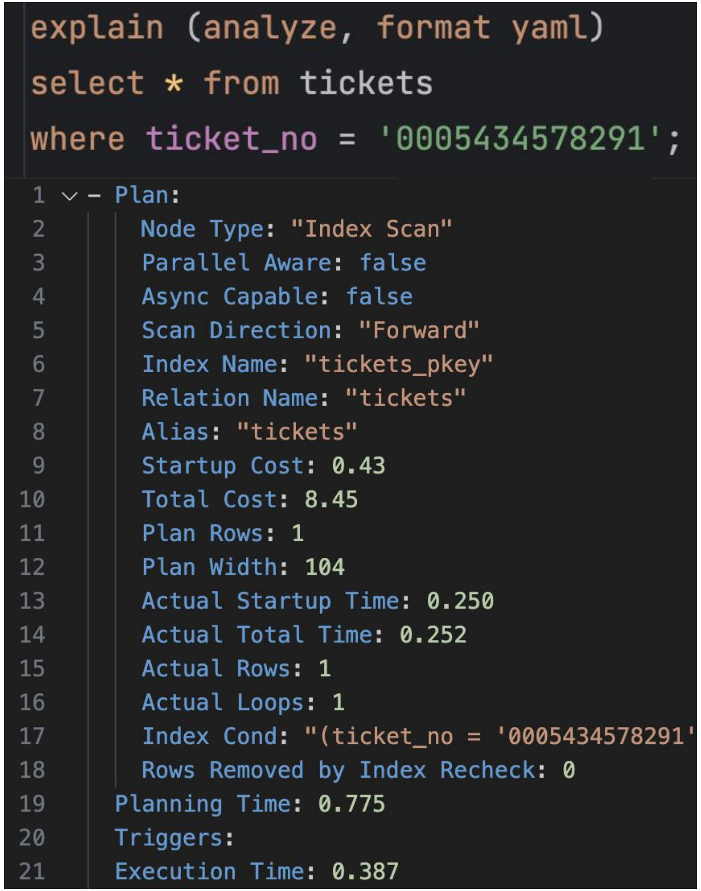


2.2.2. Types of Scanning

- **Bitmap Index Scan + Bitmap Heap Scan**: using an index to generate a bitmap of which parts of the table likely contain the matching rows, and then access the actual table to get these rows using the bitmap - this is particularly useful to combine different indexes

2.3. Reading Execution Plan

- **Node Type**: types of operation
  - Seq Scan
  - Index Scan
  - Index Only Scan
  - Bitmap Index Scan
  - Nested Loop Join
  - Sort
  - Limit
  - …
- **Index Name**: the index in used
- **Index Cond**: the condition used to scan on the index

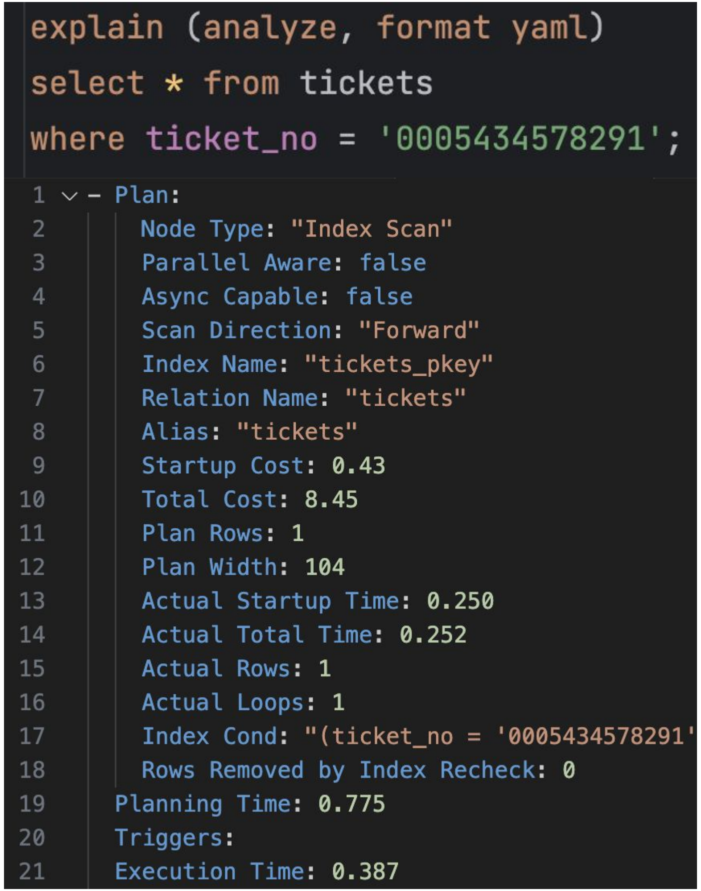


```
explain (analyze, format yaml)
select * from tickets
where ticket_no = '0005434578291';
```

```
- Plan:
    Node Type: "Index Scan"
    Parallel Aware: false
    Async Capable: false
    Scan Direction: "Forward"
    Index Name: "tickets_pkey"
    Relation Name: "tickets"
    Alias: "tickets"
    Startup Cost: 0.43
    Total Cost: 8.45
    Plan Rows: 1
    Plan Width: 104
    Actual Startup Time: 0.250
    Actual Total Time: 0.252
    Actual Rows: 1
    Actual Loops: 1
    Index Cond: "(ticket_no = '0005434578291'
    Rows Removed by Index Recheck: 0
  Planning Time: 0.775
  Triggers:
  Execution Time: 0.387
```

2.3. Reading Execution Plan

- Estimate Fields:
  - **Startup Cost**: the estimated amount of overhead necessary to start the operation (get the first record)
  - **Total Cost**: the estimated total cost of this operation and its descendants
  - **Plan Rows**: the number of rows the planner expects to be returned by the operation
  - Plan Width: the estimated average size of each row

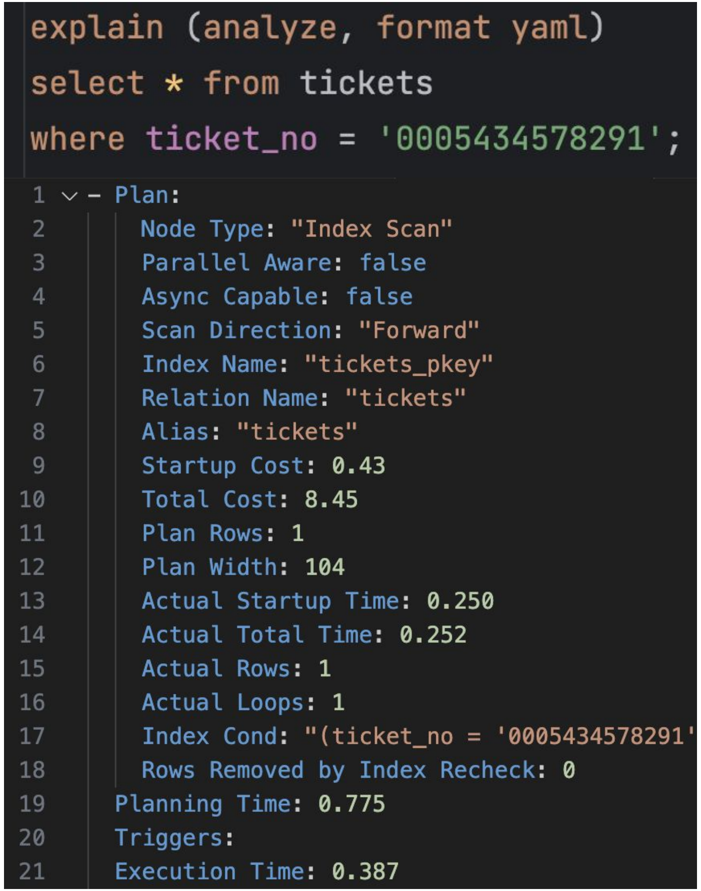


```
explain (analyze, format yaml)
select * from tickets
where ticket_no = '0005434578291';
```

```
- Plan:
    Node Type: "Index Scan"
    Parallel Aware: false
    Async Capable: false
    Scan Direction: "Forward"
    Index Name: "tickets_pkey"
    Relation Name: "tickets"
    Alias: "tickets"
    Startup Cost: 0.43
    Total Cost: 8.45
    Plan Rows: 1
    Plan Width: 104
    Actual Startup Time: 0.250
    Actual Total Time: 0.252
    Actual Rows: 1
    Actual Loops: 1
    Index Cond: "(ticket_no = '0005434578291'
    Rows Removed by Index Recheck: 0
  Planning Time: 0.775
  Triggers:
  Execution Time: 0.387
```

2.3. Reading Execution Plan

- Actual Value Fields:
  - **Actual Startup Time**: the amount of time it takes to get the first row out of the operation
  - **Actual Total Time**: the actual amount of time spent on this operation and all of its children
  - **Actual Rows**: the number of rows returned by the operation
  - Actual Loops: the number of times the operation is executed

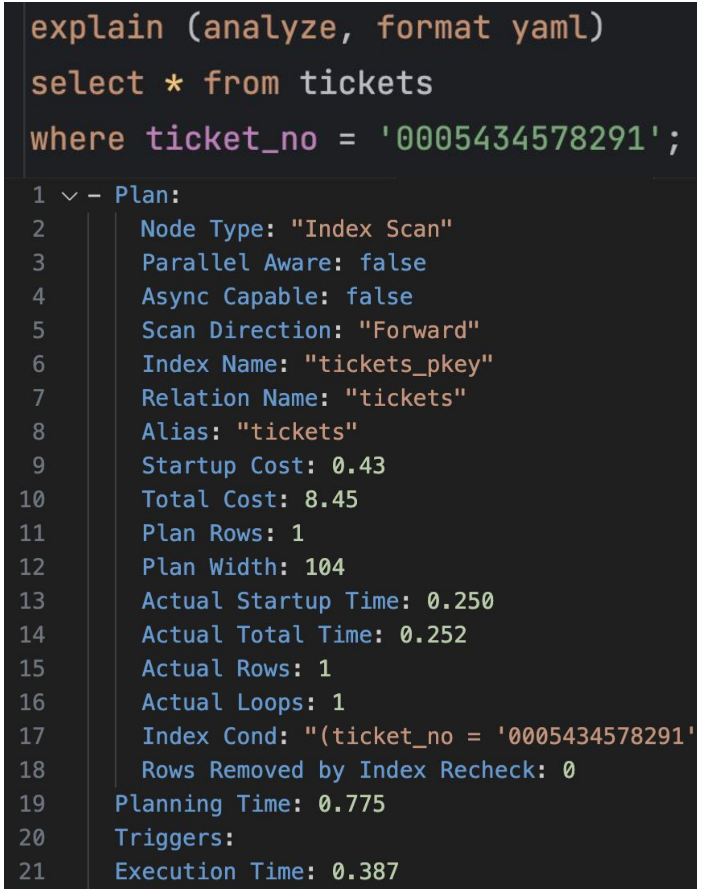


```
explain (analyze, format yaml)
select * from tickets
where ticket_no = '0005434578291';
```

```
- Plan:
    Node Type: "Index Scan"
    Parallel Aware: false
    Async Capable: false
    Scan Direction: "Forward"
    Index Name: "tickets_pkey"
    Relation Name: "tickets"
    Alias: "tickets"
    Startup Cost: 0.43
    Total Cost: 8.45
    Plan Rows: 1
    Plan Width: 104
    Actual Startup Time: 0.250
    Actual Total Time: 0.252
    Actual Rows: 1
    Actual Loops: 1
    Index Cond: "(ticket_no = '0005434578291'
    Rows Removed by Index Recheck: 0
  Planning Time: 0.775
  Triggers:
  Execution Time: 0.387
```

2.3. Reading Execution Plan

- Buffer fields:
  - **Shared Hit Blocks**: number of blocks read from cached indexes/tables



```
explain (analyze, format yaml)
select * from tickets
where ticket_no = '0005434578291';
```

```
- Plan:
    Node Type: "Index Scan"
    Parallel Aware: false
    Async Capable: false
    Scan Direction: "Forward"
    Index Name: "tickets_pkey"
    Relation Name: "tickets"
    Alias: "tickets"
    Startup Cost: 0.43
    Total Cost: 8.45
    Plan Rows: 1
    Plan Width: 104
    Actual Startup Time: 0.250
    Actual Total Time: 0.252
    Actual Rows: 1
    Actual Loops: 1
    Index Cond: "(ticket_no = '0005434578291'
    Rows Removed by Index Recheck: 0
  Planning Time: 0.775
  Triggers:
  Execution Time: 0.387
```

3. Practices

3.1. Notes before practices

- More Datasets: Kaggle
- No way to show all candidate plans, they are removed right after optimizer plans

Demo

3.3. Key Takeaways

- Btree for range query, Hash for equal query.
- Index is suitable for fetch a small number of records.
- **Use composite indexes**, Ordering matters: high cardinality first.
- Limit the number of indexes by **leveraging index condition pushdown.**
- Leverage covering index (by using the INCLUDE for PostgreSQL).
- Do not reply on framework, lib to generate SQL.
- Inspect the execution plan of the generated SQL.
- PostgreSQL and MySQL do not work the same. Practice and practice with multiple databases
- Optimization in depth using pg_stat_statemements.
  https://www.postgresql.org/docs/current/pgstatstatements.html

Recap

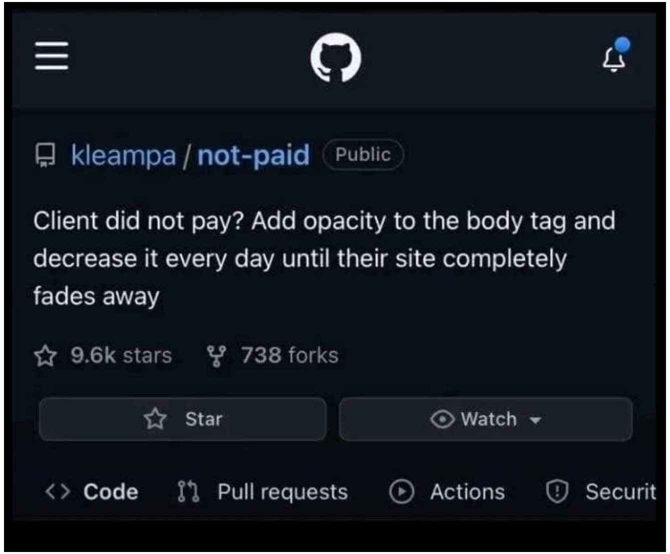
- Query Optimization includes 2 part: direct + indirect.
- Index is suitable for fetch a small result set.
- Inspect the execution plan of every production queries using EXPLAIN ANALYZE.

References

- https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-copy-table/
- https://blog.nodeswat.com/making-slow-queries-fast-with-composite-indexes-in-mysql-eb452a8d6e46
- https://www.pgmustard.com/blog/2019/9/17/postgres-execution-plans-field-glossary
- https://www.youtube.com/watch?v=Ls-uE1V31IE
- https://www.pgmustard.com/blog
- https://www.percona.com/blog/mysql-101-how-to-find-and-tune-a-slow-sql-query/

Homework

- Optimize 1 query of your project
  - Show solution
  - Explain why
- Index for the schema of booking flights



kleampa / not-paid Public

Client did not pay? Add opacity to the body tag and decrease it every day until their site completely fades away

9.6k stars 738 forks

Thank you 🙏

RONIN™
ENGINEER