Debugging Postgres Queries: Takeaways

Syntax

- Returning a query plan:

```
conn = psycopg2.connect(dbname="dq", user="hud_admin", password="abc123")
cur = conn.cursor()
cur.execute('EXPLAIN SELECT * FROM homeless_by_coc')
print(cur.fetchall())
```

- Formating the query plan of a query:

```
conn = psycopg2.connect(dbname="dq", user="hud_admin", password="abc123")
cur = conn.cursor()
cur.execute("EXPLAIN (format json) SELECT COUNT(*) FROM homeless_by_coc WHERE year >
'2012-01-01'")
```

- Returning actual run time statistics of a query:

```
conn = psycopg2.connect(dbname="dq", user="hud_admin", password="abc123")
cur = conn.cursor()
cur.execute("EXPLAIN (ANALYZE, FORMAT json) SELECT COUNT(*) FROM homeless_by_coc WHERE
year > '2012-01-01'")
```

- Reverting any changes made to the database:

```
import psycopg2
conn = psycopg2.connect(dbname="dq", user="hud_admin", password="abc123")
cur = conn.cursor()
 # Modifying change to the database.
cur.execute("EXPLAIN ANALYZE ALTER TABLE state_info ADD COLUMN notes text")
 # Reverting the change.
conn.rollback()
```

- Executing an inner join in Postgres:

  ```
  conn = psycopg2.connect(dbname="dq", user="hud_admin", password="abc123")

  cur = conn.cursor()

  cur.execute("EXPLAIN (ANALYZE, FORMAT json) SELECT hbc.state, hbc, coc_number,

  hbc.coc_name, si.name FROM homeless_by_coc as hbc, state_info as si WHERE hbc.state =

  si.postal")
  ```

Concepts

- Path of a query:
  - The query is parsed for correct syntax. If there are any errors, the query does not execute and you receive an error message. If error-free, then the query is transformed into a query tree.
  - A rewrite system takes the query tree and checks against the system catalog internal tables for any special rules. Then, if there are any rules, it rewrites them into the query tree.
  - The rewritten query tree is then processed by the planner/optimizer which creates a query plan to send to the executor. The planner ensures that this is the fastest possible route for query execution.
  - The executor takes in the query plan, runs each step, then returns back any rows it found.
- The `EXPLAIN` command examines the query at the third step in the path.
- For any query, there are multiple paths leading to the same answer and the paths keep increasing as the complexity of a query grows.
- Query plans are a `Seq Scan` , which means the executor will loop through every row one at a time.
- You can format the query plan in the following formats:
  - Text
  - XML
  - JSON
  - YAML

- Both `Startup Cost` and `Total Cost` are estimated values that are measured as an arbitrary unit of time.
    - `Startup Cost` represents the time it takes before a row can be returned.
    - `Total` Cost includes `Startup Cost` and is the total time it takes to run the node plan until completion.
- Joins are computationally expensive to perform and the biggest culprit in delaying execution time.
- Before a join can occur, a `Seq Scan` is performed on each joined table. These operations can quickly become inefficient as the size of the tables increase.

Resources

- Postgres EXPLAIN statement
- Full table scan