PostgreSQL Cheat Sheet

(source: https://www.udemy.com/course/sql-and-postgresql)

V2021.03.14

(Dr Yan Xu)

Basic SQL

- SELECT:
  - ***select*** *column1, column2, ...*
  - ***from*** *table_name*
  - ***where*** *condition;*
- INSERT:
  - ***insert into*** table_name (column1, column2, ...)
  - ***values*** (value1, value2, ...);
- UPDATE:
  - ***update*** table_name
  - ***set*** column1 = value1, column2 = value2, ...
  - ***where*** condition;
- DELETE:
  - ***delete from*** table_name
  - ***where*** condition;

Intermedia SQL

- GROUP BY / ORDER BY
- LIMIT / OFFSET
- SET OPERATIONS:
  - ***union & union all***
  - ***intersect & intersect all***
- JOIN
  - ***left join / right join***
  - ***inner join (default) / outer join***
- IN, BETWEEN
- WINDOW Func
  - OVER (PARTITION BY xx ORDER BY xx DESC)
  - examples:
    - row_number(), rank()
    - Lag() - before the current row
    - Lead() - after the current row
- AS - table / feature rename

Scheme Design

- Entity Relationship Diagram
- Entities?
- Relationship?
  - 1-to-1:
  - 1-to-N / N-to-1: forign key
  - N-to-N: a separate relationship table
- Primary Key (e.g. auto-generated ID or uuid)
- Forign key & Deletion Protection (careful-to-use)
- Common Data Types:
  - integer, double precision, numeric
  - char, varchar, varchar(N), text
  - boolean, date, time, datetime, interval
- Common attributes to consider:
  - created_at, updated_at, status
- Extra considerations:
  - NOT NULL
  - DEFAULT
  - UNIQUE
- Criterion
  - clear logic
  - query performance
  - less duplication

Scheme Management

- CREATE:
  - ***create table*** *users (*
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    price NUMERIC(10,2) DEFAULT 0
    );
  - ***create index*** *idx_name ON users(name);*
  - ***create view*** *v(c1,c2) AS*
    SELECT c1, c2
    FROM users;
- DROP:
  - ***drop table*** *users*
  - ***drop index*** *idx_name*
- ALTER:
  - ***alter table*** *users* ***add*** *column*
  - ***alter table*** *users* ***add*** constraint
  - ***alter table*** *users* ***rename to*** another_name

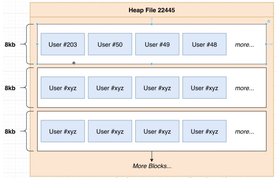
Database-side Verification & Constraints

- DB-side vs App-side Verification
- Common Verification:
  - NOT NULL
  - DEFAULT
  - UNIQUE
  - MULTI-COLUMN UNIQUE
  - CHECK
- Example:

```
CREATE TABLE USERS (
    id SERIAL PRIMARY KEY,
    ssn VARCHAR(10) UNIQUE,
    name VARCHAR(30) NOT NULL,
    weight INTEGER CHECK(weight > 0)
);
```

PostgreSQL Internal

- Data Location
  - show data_directory;
- Data Storage
  - Heap file
  - File block
  - Records

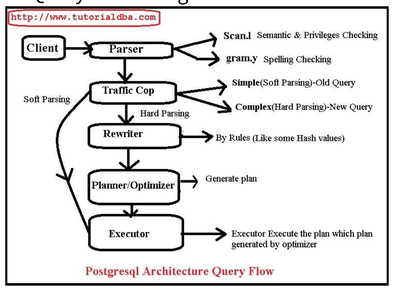


(table users is just a heap file in disk)

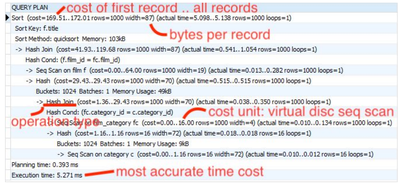
- Index
  - Index is a carefully designed B+ Tree as a smaller table (and thus a smaller heap file)
  - Postgresql automatically creates Index for Primary key and Unique columns.
- Metadata in DB
  - pg_database
  - pg_class: storage metadata
  - pg_stat_activity: tracing & blame metadata
  - pg_stat_*: used for query planning

Query Planning and Cost Analysis

- Query Processing Framework



- Commands:
  - Explain
  - Explain Analyze
- Query Plan



- Planner Cost Constants
  - postgresql pre-defines the costs of most operations (e.g. an index read) relative to a sequential disc scan
  - check details here
- Query Planning relies on *pg_stats* (*it is a view*) for Cost Estimation
- Command to update statistics:
  - ANALYZE
  - ANALYZE TABLE_NAME
  - ANALYZE TABLE_NAME (COLUMN_NAME)
- Scanning large tables should be avoided if possible, by using indexes

Common Table Expression (with ...)

- ***with*** should be used to simplify SQL
- Format:

```
with engineers as (
    select *
    from employees
    where dept='Engineering'
)
select *
from engineers
where ...
```

(WITH, CTE name, CTE Body, CTE Usage)

- ***with*** does not impact performance

Recursive Common Table Expression

- It should not be used
  - Problem: could easily be very costly
- Format:

```
with recursive ancestors as (
    select * from folks
    where name = 'Alex'
    union [all]
    select f.*
    from folks as f, ancestors AS a
    where
      f.id = a.father or f.id = a.mother
)
select * from ancestors;
```

("recursive", Anchor part, Recursive use of CTE, Recursive part)

Transaction

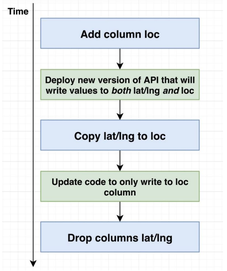
- All or Nothing
  - The impact is within a local env before COMMIT
  - It is similar with local branch in Git
- Format:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts
    WHERE name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts
    WHERE name = 'Bob');
COMMIT;
```

(One transaction)

Schema / Data Migration Plan

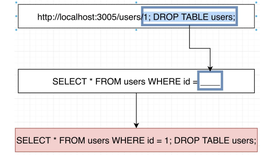
- Any schema/data migration needs to have a PLAN, no matter how small the change is (e.g. column name change).
- Schema/Data migration may take days
- Migration Plan Example



- Schema Migration as Code (*Optional*)
  - python: yoyo-migrations

Security

- SQL Injection Risk



- Prepared Statement (enforce single statement)