Views and Functions

Advanced SQL and Cloud

Objectives

1. Explain what a view is and how it can simplify writing and encapsulate logic.
2. Differentiate between standard views and materialized views.
3. Create and use views to organize repeated logic and present analyst-friendly tables.
4. Understand what a function is and how it differs from a view or simple query.

Introduction: Views and Stored Procedures

Analysts often repeat similar queries across projects.

Views and functions encapsulate logic, leading to less repetition and more consistency. Think: functions from Python.

Both tools support reproducibility and easier collaboration.

Recall: Temporary Tables

A temporary table (temp table) is a table that you create during your session.

They can be useful for storing a result set and being able to reference that result set again without having to rerun the query that created it.

However, they have the disadvantage of disappearing at the end of a session.

Views and Materialized Views

A major downside to a temp table is that it disappears after the session ends. For repeated dashboards or reports, a **materialized view** might be a better option.

A **view** is a saved query in the database.

- Acts like a virtual table; you can SELECT from it just like any other table.

A standard view will re-run the query every time, but a **materialized view** is stores the results in the database so that it doesn't have to be run each time.

- Materialized views are much faster for repeated use, but must be refreshed if the data is updated.
- Think: snapshot of query results at one point in time.

Views

What?

- Saved queries stored in the database

Why?

- Abstract complexity (long joins, business logic)
- Enforce consistency across reports and dashboards
  - Eg. enforce standard definitions of KPIs/metrics
- Limit exposure to sensitive columns (security)
- Easily integrate with external tools like Tableau, Power BI, or Python, providing a cleaned/prepared dataset layer for further analysis

Views

To create a regular view:

```
CREATE VIEW recent_orders AS
SELECT *
FROM orders
WHERE order_date > CURRENT_DATE - INTERVAL '90 days';
```

Materialized Views

To create a materialized view:

```
CREATE MATERIALIZED VIEW recent_orders_mv AS
SELECT *
FROM orders
WHERE order_date > CURRENT_DATE - INTERVAL '90 days';
```

To refresh a materialized view:

```
REFRESH MATERIALIZED VIEW recent_orders_mv;
```

Materialized views are often refreshed on a schedule.

Views - Standard vs. Materialized

Standard

- Logical layer only; no data stored
- Evaluated at query time
- Always up to date
- Can be slow on large/complex joins

Materialized

- Persist data
- Since they results are precomputed, the are often faster
- Can be stale until refreshed, so important to know the refresh cadence
- Can be indexed
- Need refreshing

Table-Returning Functions

What?

- A function that returns rows like a table

Why?

- Encapsulate reusable queries with parameters

Table-Returning Functions

Example:

```
CREATE FUNCTION patient_visit_summary_fn(min_date DATE, max_date DATE)
RETURNS TABLE(person_id INT, num_visits INT)
LANGUAGE SQL
AS $$
  SELECT person_id, COUNT(*) AS num_visits
  FROM visit_occurrence
  WHERE visit_start_date BETWEEN min_date AND max_date
  GROUP BY person_id;
$$;
```

Table-Returning Functions

To use this function, you can treat is just like any other table.

```
SELECT *
FROM patient_visit_summary_fn('01-01-2009', '02-01-2009');
```

or

```
SELECT *
FROM patient_visit_summary_fn('01-01-2009', '02-01-2009')
WHERE num_visits >= 20;
```