SQL Level 3

Part 2

noble desktop

Views

What is a View?

- A view is a “virtual table” that allows you to query the data in it. A view contains rows and columns, just like a real table.
- The columns in a view are columns from one or more real tables.
- Views do not store data; the data resides in the actual table(s).
- You can use SQL functions, WHERE, and JOIN statements in a view and present the data as if the data were coming from one single table.
- A view always shows up-to-date data. The database engine recreates the data, using the view’s SQL statement, every time a user queries a view.

Advantages of a View

- You can use a view to hide the complexity of underlying tables to end-users and external applications.
- A view can limit access when needed. You may not want sensitive data to be queryable by all users. A view can expose only the non-sensitive data to users who are granted access.
- So instead of being given access to tables, you may only be given access to views.

Disadvantages of a View

- Performance: Querying data from a view can be slow, especially if the view is based on other views.
- SQL Server can create Indexed Views, which can improve performance. You query them the same as a regular view. Learn more about creating them.
- Tables Dependency: Whenever you change the structure of the tables used by a view, you have to change the view as well.

Creating a View

```
CREATE VIEW my_view AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

Querying a View

```
SELECT *
FROM my_view
ORDER BY column1;
```

PostgreSQL Only: Materialized Views

- Regular **Views** query the live database so they always show up-to-date data.
- **Materialized Views** cache the results for better performance.
- When querying materialized views, it will use the cached results, not the live database. So they must be updated if you need to work with data that's changed *after* the materialized view was created.

Exercise

Open the file “2.0 Views.sql” (in SQL Level 3 folder)

Variables

Variables

- Variables let you store something to use one or more times.
- They can be used to make your code easier to read, or to make it easier to update.
- First you must declare a variable and its data type.
  What will you put in it? Text, numbers, dates, etc.
- Later you can use the variable by the name you give it.

Variables

In PostgreSQL we use variables with functions, not by themselves as you can in SQL Server.

Exercise

SQL Server Only: Open the file “2.1 Variables.sql” (in SQL Level 3 folder)

User-Defined Functions

User-Defined Functions

- Functions do something and return a value.
- We've seen SQL-provided functions, but you can define your own.
- Used-Defined Functions (UDF) are created by you (or someone else) and stored in the database.
- Parameters let you alter the function each time you run it.
- Parameters are essentially variables within a function.
- Functions may use parameters, but they do not have to.
  Examples: NOW( ) or GETDATE( ) have no parameters to change how they work. ROUND( ) does have parameters.

Functions vs Views

- You cannot declare parameters for a view.
- When you query a view, it will always return the same data, although you can then filter, aggregate, etc. that data.
- You can declare parameters for a function, which can act like a view that you can customize each time you run it.

Types of Functions

- There are *scalar-valued* and *table-valued* functions.
- **Scalar functions** return a single value (called a scalar value).
- **Table functions** return a table.
- ROUND( ) and LEN( ) or LENGTH( ) are examples of built-in scalar functions because they return a single value.
- A user-defined *scalar function* has zero, one, or multiple parameters and returns a single value.
- A user-defined *table function* has zero, one, or multiple parameters and returns a table.

Exercise

Open the file “2.2 Functions in PostgreSQL.sql” or
“2.2 Functions in SQL Server” as appropriate
(in SQL Level 3 folder)

Stored Procedures

Mainly for SQL Server

Stored Procedures in SQL Server

- Stored procedures are generally reserved for automating various system processes (including database modifications), but they can also do simpler tasks.
- SQL Server has system-provided stored procedures, but you can define your own.
- Stored procedures cannot be used in a SELECT statement, so they cannot be used as a building block in a larger query. Views and functions are preferred in those situations.

Postgres Stored Procedures vs. Functions

As of Postgres 13, returning from a PROCEDURE is still very limited.

A FUNCTION offers more options to return values, doesn’t need to be run separately with CALL, and can be integrated in bigger queries. Chances are, that’s what you wanted in the first place, and you were just being mislead by the widespread misnomer “stored procedure”.

The rule of thumb: if you don’t need to manage transactions from within, you probably want to use a function instead of a procedure. Later, Postgres procedures may be extended to be able and return multiple result sets (per SQL standard), but not yet.

Source: Stack Overflow

Exercise

SQL Server Only: Open the file “2.3 SQL Server Stored Procedures.sql” (in SQL Level 3 folder)

More Learning Resources

Learning Resources

- For more practice: sqlcourse.com
- Reference: dataschool.com/learn-sql
- SQL Commands Cheat Sheet (read on the page, or download the cheat sheet PDF)