Google Cloud

SQL strategies to join data effectively

Previously, you learned that in order to efficiently query the data they need, data professionals can use **JOINs** to combine tables. In this reading, you'll explore some strategies to help you join data more effectively, including avoiding anti-patterns, the difference between join and set operators, and how to handle NULL values in joins.

Anti-patterns

In SQL, an anti-pattern is a code solution that seems functional, but actually has unintended or unwanted effects. These effects can include performance and memory issues, data loss, and unexpected query results. Also, these anti-pattern solutions can create a computation burden on your system, making querying slow and resource intensive. These anti-patterns can become so common that you don't realize they're a mistake. There are two **JOIN** anti-patterns in particular that you should avoid:

- **Self join anti-pattern:** A self join is when a table is joined with itself. For example, you could use a self join on a table where first and last name data is stored in separate columns to create another version of the table where those columns are merged. On the surface, this can seem like a good solution for editing tables. But in reality, self joins can be resource intensive and slow. Plus, the syntax gets complicated and hard to read pretty quickly. Instead, it's usually better to use a window function instead.
- **Cross join anti-pattern:** A cross join is when each row from the first table in a query is joined to every row in the second table. This can create an output where the number of rows in the left table is multiplied by the number of rows in the right table. Sometimes, these queries can't even finish executing. To avoid this anti-pattern, you should pre-aggregate the data before joining, or use a window function instead.

JOINs vs set operators

When combining tables, **JOINs** are a common function data professionals can use. But there are also set operators that can also be useful. Set operators are specifically used for tables that already have the same fields. **JOINs** are typically best for tables with distinct fields that are connected by keys. For example, some useful set operators include:

- **UNION:** The **UNION** function can combine the outputs of two different **SELECT** statements into one table. For example, if you're working with a table of customer information that includes names, addresses, and contact information, you can use a **UNION** statement to connect it to another table with employee information that also

includes names, addresses, and contact information.

- **INTERSECT:** The **INTERSECT** statement can be used to pull information from two tables that have identical information. For example, if one table includes both order information and data about suppliers, and a second table includes data about suppliers, you can use **INTERSECT** to find the intersections of the two tables.
- **EXCEPT: EXCEPT** is like the opposite of **UNION**; instead of connecting tables where they're identical, it finds the unique records from the first table. For example, imagine you have two tables that include employee data. The first table contains employee names and upcoming vacation days. The second table includes employees name, contact information, and department. When you use **EXCEPT** on these tables, it produces an output with a list of employees with upcoming vacation days, along with their information.

How to handle NULL values

When using **JOINs**, NULL values can present challenges. If the tables you're joining have mismatched rows, then your resulting table will have NULLs. Or, if your existing tables have NULLs, your query might treat them as zeroes, skewing your resulting table. One way to manage this is to deal with NULL values before you use **JOINs**. For example, the **COALESCE** function is used to identify non-NULL values in a column. But it can also be used to replace NULL values with a default value in order to more easily and accurately join tables.

Key takeaways

Using **JOINs** or finding other ways to join data effectively is a key part of the data transformation process. It's a skill you can learn and develop over time that will be a large part of your role as a cloud data professional. Ensuring that you're using **JOINs** effectively will also help you avoid over-burdening your system, and wasting time, resources, and storage.