Denormalization

by Sophia

WHAT'S COVERED

This tutorial explores the need to denormalize a database in three parts:

1. Need For Denormalization
2. Invoice Example
3. Analytical Databases

1. Need For Denormalization

Although you have mostly focused on normalization to the third normal form (3NF), there are instances when you may want to denormalize a database. Although an optimal transactional database should be at least in 3NF, the further you normalize, the more tables are created. To generate useful information from the database, you have to join the tables together. The more joins you have, the more input/output operations and processing is required. Although most databases can handle this processing quite effectively, is can become more of a challenge for databases that are much larger in size.

There are also some data anomalies that may not make sense to split off.

> EXAMPLE For example, in the US, the zip code can define the city and state which, through normalization rules, you may have split off in a separate reference table. However, would it make more sense to avoid the redundancy of the city and state?

In some situations, it may make sense, but it may not in others. Looking at our database in PostgreSQL, you see that the city/state/zip combination exists in the invoice, customer, and employee tables. Keeping it in a single table would introduce some redundant data in the data model. This is an example where you may want to avoid those extra join conditions.

2. Invoice Example

You can also have situations where you want to store pre-aggregated data or derived data. The invoice table with the total is a good example of this. The total could be calculated for the invoice by adding the total of the quantity and unit_price of each row from the invoice. However, if you needed to use the invoice total in different situations, you would have to calculate it each time, which may not make sense. You could also have a temporary denormalized table with data stored in a format where you may have repeating groups of columns. This type of query may be impossible to generate the data required purely through SQL and may need other programming languages to store that information.

3. Analytical Databases

With data warehouses for business intelligence, you may also have denormalized data since the data has gone through a transactional database. For analytical purposes, you are more worried about performance than data redundancy or data anomalies. If you do not need to worry about the constant data insert, update or deletion, denormalization will not be an issue.

SUMMARY

In this tutorial, you learned that there can be a **need for denormalization** to help lower redundancy. You learned using the **invoice example** how a temporary denormalized table could help in different situations. Finally, you learned regarding **analytical databases**, if you are not as worried that data redundancies or anomalies will occur and are more worried about performance, having denormalized data that has gone through a transactional database may not be an issue.

Source: Authored by Vincent Tran