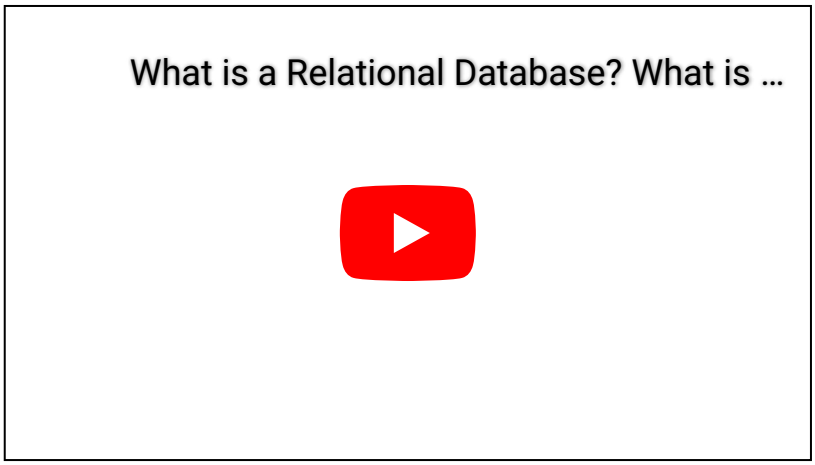
A

What is denormalization and how does it work?

Denormalization is the process of adding precomputed redundant data to an otherwise normalized relational database to improve read performance of the database. Normalizing a database involves removing redundancy so only a single copy exists of each piece of information. Denormalizing a database requires data has first been normalized.

With denormalization, the database administrator selectively adds back specific instances of redundant data *after* the data structure has been normalized. A denormalized database should not be confused with a database that has never been normalized.

Using normalization in SQL, a database will store different but related types of data in separate logical tables, called relations. When a query combines data from multiple tables into a single result table, it is called a join. The performance of such a join in the face of complex queries is often the occasion for the administrator to explore the denormalization alternative.



Normalization vs. denormalization

Denormalization addresses a fundamental fact in databases: Read and join operations are slow.

In a fully normalized database, each piece of data is stored only once, generally in separate tables, with a relation to one another. For this information to become useable it must be read out from the individual tables, as a query, and then joined together. If this process involves large amounts of data or needs to be done many times a second, it can quickly overwhelm the hardware of the database and slow performance -- or even crash the database.

As an example, imagine a fruit seller has a daily list of what fruit is in stock in their stand and a daily list of the market prices for all fruits and vegetables. This would be two separate tables in a normalized database. If a customer wanted to know the price of an item, the seller would need to check both lists to determine if it is in stock and at what price. This would be slow and annoying.

Therefore, every morning, the seller creates another list with just the items in stock and the daily price, combining the two lists as a quick reference to use throughout the day. This would be a denormalized table for speeding up reading the data.

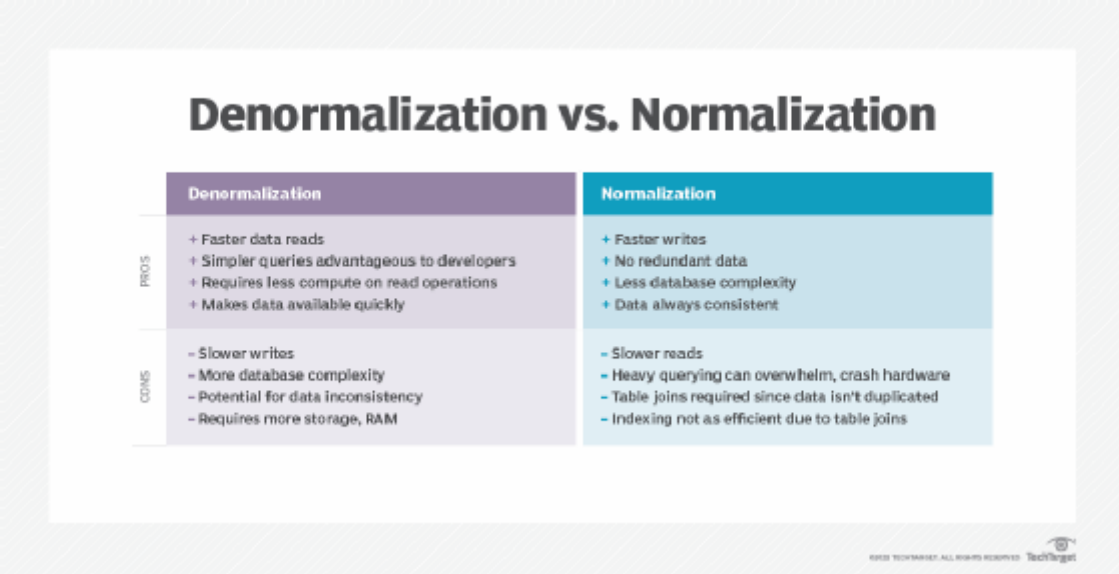
An important consideration for normalizing data is if the data will be read heavy or write heavy. Because data is duplicated in a denormalized database, when data needs to be added or modified, several tables will need to be changed. This results in slower write operations.

Therefore, the fundamental tradeoff becomes fast writes and slow reads in normalized databases versus slow writes and fast reads in denormalized.

For example, imagine a database of customer orders from a website. If customers place many orders every second but each order is only read out a few times during order processing, prioritizing write performance may be more important (a normalized database). On the other hand, if each order is read out hundreds of times per second to provide a 'based on your order recommendations' list or is read by big data trending systems, then faster read performance will become important (a denormalized database).

Another important consideration in a denormalized system is data consistency. In a normalized database, each piece of data is stored in one place; therefore, the data will always be consistent and will never produce contradictory results. Since data may be duplicated in a denormalized database, it is possible that one piece of data is updated while another duplicated location is not, which will result in a data inconsistency called an update anomaly. This places extra responsibility on the application or database system to maintain the data and handle these errors.

Denormalization has become commonplace in database design. Advancing technology is addressing many of the issues presented by denormalization, while the decrease in cost of both disk and RAM storage has reduced the impact of storing redundant data for denormalized databases. Additionally, increased emphasis on read performance and making data quickly available has necessitated the use of denormalization in many databases.



Denormalization addresses the slow read and join operations of normalized databases, and is increasingly becoming more common. Both kinds have the pluses and minuses, however.

Denormalization pros and cons

Performing denormalization on databases has its pros and cons, including the following:

Denormalization pros

- Faster reads for denormalized data
- Simpler queries for application developers
- Less compute on read operations

Denormalization logical design

Database administrators can perform a denormalization as a built-in function of a database management system (DBMS), or they can introduce it as part of an overall database design. If implemented as a DBMS feature, the database itself will handle the denormalization and keep the data consistent. If a custom implementation is used, the database administrator and application programs are responsible for data consistency.

The specifics of the automated denormalization system will vary between DBMS vendors. Since denormalization is complicated,

automated denormalized views are generally only a feature of a paid DBMS. Microsoft SQL Server uses indexed views for denormalized data, for example, while Oracle databases call precomputed tables materialized views. Both use cost-based analyzers to determine if a prebuilt view is needed.

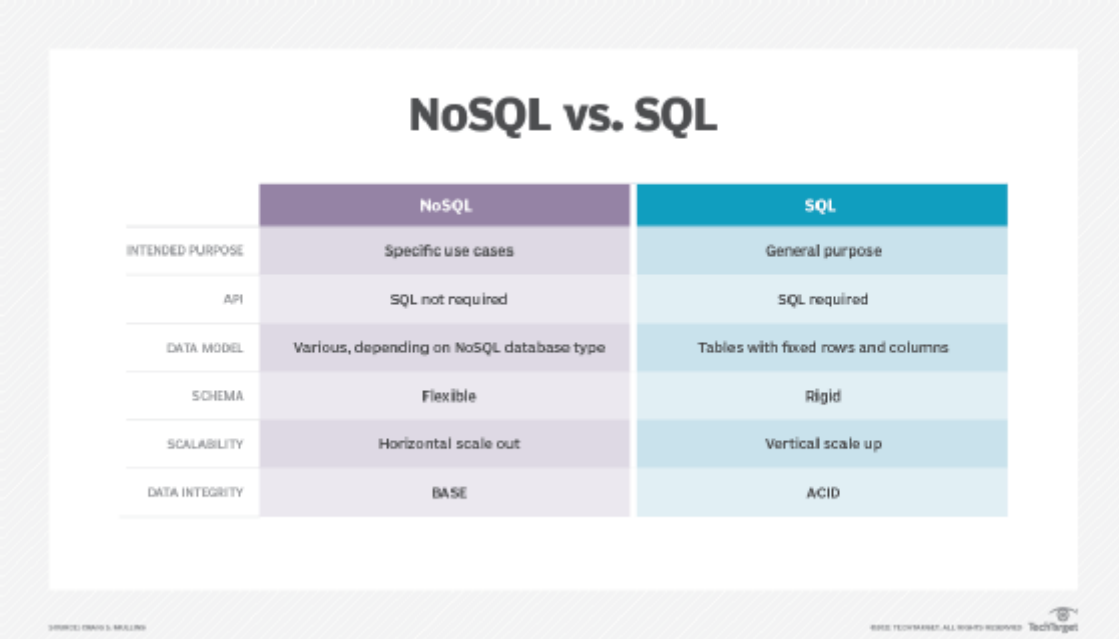
Users can add denormalized tables as part of the database architecture design as well. MySQL can do this with a create view statement, for instance.

Denormalization in data warehousing and NoSQL databases

Denormalization often plays an important role in relational data warehouses. Because data warehouses contain massive data sets and may host many concurrent connections, optimizing read performance and minimizing expensive join operations is important. This is particularly true in dimensional databases as prescribed by influential data warehouse architect and author Ralph Kimball.

Kimball's emphasis on dimensional structures that use denormalization is intended to speed query execution, which can be especially important in data warehouses used for business intelligence. Denormalization helps data warehouse administrators ensure more predictable read performance.

Examples of denormalization go beyond relational and SQL. Applications based on NoSQL databases often employ this technique -- particularly document-oriented NoSQL databases.



Denormalization has a place with SQL and NoSQL databases, as well as in data warehousing.

Such databases often underlie content management systems for web profile pages that benefit from read optimizations. The goal of denormalization in this context is to reduce the amount of time needed to assemble pages that use data from different sources. In such cases, maintaining data consistency becomes the job of the application and, in turn, the application developer.

Columnar databases such as Apache Cassandra also benefit greatly from denormalized views, as they can use high compression to offset higher disk usage and are designed for high read access.

This was last updated in May 2021