WIKIPEDIA

# Denormalization

**Denormalization** is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data.[1][2] It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations. Denormalization should not be confused with Unnormalized form. Databases/tables must first be normalized to efficiently denormalize them.

A normalized design will often "store" different but related pieces of information in separate logical tables (called relations). If these relations are stored physically as separate disk files, completing a database query that draws information from several relations (a *join operation*) can be slow. If many relations are joined, it may be prohibitively slow. There are two strategies for dealing with this. One method is to keep the logical design normalized, but allow the database management system (DBMS) to store additional redundant information on disk to optimise query response. In this case it is the DBMS software's responsibility to ensure that any redundant copies are kept consistent. This method is often implemented in SQL as indexed views (Microsoft SQL Server) or materialised views (Oracle, PostgreSQL). A view may, among other factors, represent information in a format convenient for querying, and the index ensures that queries against the view are optimised physically.

Another approach is to denormalize the logical data design. With care this can achieve a similar improvement in query response, but at a cost—it is now the database designer's responsibility to ensure that the denormalized database does not become inconsistent. This is done by creating rules in the database called *constraints*, that specify how the redundant copies of information must be kept synchronised, which may easily make the de-normalization procedure pointless. It is the increase in logical complexity of the database design and the added complexity of the additional constraints that make this approach hazardous. Moreover, constraints introduce a trade-off, speeding up reads (`SELECT` in SQL) while slowing down writes (`INSERT`, `UPDATE`, and `DELETE`). This means a denormalized database under heavy write load may offer *worse* performance than its functionally equivalent normalized counterpart.

A denormalized data model is not the same as a data model that has not been normalized, and denormalization should only take place after a satisfactory level of normalization has taken place and that any required constraints and/or rules have been created to deal with the inherent anomalies in the design. For example, all the relations are in third normal form and any relations with join and multi-valued dependencies are handled appropriately.

Examples of denormalization techniques include:

- "Storing" the count of the "many" elements in a one-to-many relationship as an attribute of the "one" relation
  - Adding attributes to a relation from another relation with which it will be joined
- Star schemas, which are also known as fact-dimension models and have been extended to snowflake schemas
- Prebuilt summarisation or OLAP cubes

Denormalization techniques are often used to improve the scalability of Web applications.[3]

With the continued dramatic increase in all three of storage, processing power and bandwidth, on all levels, denormalization in databases has moved from being an unusual or extension technique, to the commonplace, or even the norm. For example, one specific downside of denormalization was, simply, that it "uses more storage" (that is to say, literally more columns in a database). In all but the most enormous imaginable systems, this particular aspect has been made irrelevant; the danger of using more storage is a non-issue.

## See also

- Cache (computing)
- Normalization
- Scalability

## References

1. G. L. Sanders and S. K. Shin. Denormalization effects on performance of RDBMS (https://pdfs.semanticscholar.org/2c79/069c01ba8d598f32e61fe367ef6d261a0cb4.pdf). In Proceedings of the HICSS Conference, January 2001.
2. S. K. Shin and G. L. Sanders. Denormalization strategies for data retrieval from data warehouses (http://portal.acm.org/citation.cfm?id=1217757). Decision Support Systems, 42(1):267-282, October 2006.
3. Z. Wei, J. Dejun, G. Pierre, C.-H. Chi and M. van Steen. Service-Oriented Data Denormalization for Scalable Web Applications (http://www.globule.org/publi/SODDSWA_www2008.html). In Proceedings of the International World-Wide Web conference, April 2008.