PURISOLVE
experience : integrity : solutions
info@purisolve.com
260 Peachtree St.
Suite 2100
Atlanta, GA 30303
A Technology-Advisory Company

Abnormal Normalization

Override database over-normalization with System Design

Database efficiency driven by Data Modeling and System Design

Efficiency and quality-performance of the transactional database is typically achieved through normalization of the database. However, when over-normalized, the result is dramatic database seizures, sluggishness, and non-response, resulting in increased redesign costs.

Need for performance limits transactional database design

The focus of this technical paper is for business owners and their database administrators to take cognizance of nuances of transactional databases and optimize its stability by aligning the database with sustainable system design, rather than over-normalization.

Need for performance limits transactional database design

While Data Modeling offers the scope of designing a database that can incorporate features for future use, most designers are compelled, by the costs of development and the need to achieve quick performance, to compromise and conceive neutral data models. These models meet immediate user requirements at minimal cost and perform on real-time basis.

What is a transactional database?

A critical phase of a database is its ability to handle a transaction. Typically, a transaction is an event of work performed on a database management system – which necessarily has to be performed atomically, entirely, consistently and cause no effect when durably stored.

This means integrity of the data is paramount and the database has to handle independent work events on a single transaction of data storage.

For example, if you purchase software for your business, a transactional double-entry should include your credit to your banking account as well as debit to your software vendor. Your database becomes a transactional database when your database can be rolled back to rewrite a transaction that was not completed due to a power failure or loss of internet connection. To achieve maximum integrity and transactional instances, the database has to be maintained through several processes. One such process is normalization.

Why are databases normalized?

A database is a group of logical units called fields under a common category or table with an established relation between them so that dependency and redundancy are minimized.

The purpose of normalization is to overcome anomalies arising out of insertion, deletion and updates

The further division of these tables into smaller tables, so as to lower redundancy, and establish the relationship between them is called normalization. This is done to improve the scope for the increased instances of modification, addition as well as deletion of fields in a given table, to eventually map to the defined relationships.

Historically, Edgar.F.Codd evolved this method of relational model normalization in 1970. Later theorists also contributed and as a result One Normal Form (1NF) to Six Normal Form (6NF) is currently in use.

Six forms of normalization

The Six forms of normalization are

- **1st Normal Form**
- **2nd Normal Form**
- **3rd Normal Form**
- **Boyce-Codd Normal form(BCNF)**
- **4th Normal Form**
- **5th Normal Form**
- **Domain Key Normal Form**

Recommended levels of Normalization

Though, it is recommended that relation normalization is best until third normal form, the ease of normalization has resulted in over-normalization that results in several unnatural scenarios of database operations.

While the standard practice is for the database design to build in capabilities for complete normalization; most administrators do opt for de-normalization for specific performance requirements, following a normalization process. This is recommended to lower the instances of Joins in queries to a manageable level.

What is over-normalization?

In a transactional database, over-normalization occurs when there are far too many JOINs used to access data; to the extent that performance is penalized and database enters deadlocks.

Over-normalization occurs when there are far too many JOINs

Typically, huge applications will quickly implement de-normalization to achieve rapid scaling back to normal performance. However, for small and medium businesses, normalization occurs when data duplication is to be avoided as business needs alter extensively over a period of time.

Therefore, it is common to reach 3NF and its alternate form BCNF at the most and not go beyond it as the trade-off topples the balanced database. Wherever excessive Normal Forms are reached, there is a trade-off with relation to speed and performance of certain applications you may be using.

Why should system design override normalization?

Due to over normal forms, database tables become top-heavy and a liability and need add-up processes such as: quick de-normalizing of tables into caching tables, or adding of new database schema to keep the database primed. All this leads to loss of efficiency, storage and of course increased costs due to normal forms that simply exist because of the over normalization.

Normalization is recommended only when it is relevant and achieves desired requirements and optimizes performance. If normalization is below level, then it leads to repetition and overburdening.

System Design should override

While over-normalization causes the greatest harm, an extensive and un-natural number of JOINs across innumerable tables. Therefore, over-normalization is not recommended and instead allowing integrated and sustainable system design to override offers better ROI on the transactional database.

Take Away

Data Modeling and System Design bring intrinsic value to the database. It is better to spend time on robust database model and design rather than slave hours away on non-productive maintenance which will finally result in redesigning. System Design should override transactional database maintenance and not over-normalization.