Daniel Marzari 450

CIS 221

Michael Sabal

3/28/20

Exam 2 Essay

It seems to be a highly contested thing to accurately define database normalization, nevertheless an overwhelming majority seem to agree with E. F. Codd, the father of relational databases and normalization, who defines it as a procedure of simple elimination which removes non-simple domains from relations (Carpenter 379). While this definition is filled with jargon, it gets at the heart of database normalization: an attempt to properly structure data relations according to their most atomic form (Carpenter 380). Writers on this topic draw attention to "non-simple relations", "unnormalized relations", and "anomalies" all of which refer to nonatomic fields - any field that extends even slightly beyond the scope of its purpose. For example, if a table was made with a column for phone numbers which could also be left blank, the ambiguity of the field is non-simple. The field's purpose could be defined as "Person P has phone number PN **OR** person P's phone number is unknown **OR** person P doesn't have a phone number" (Date 498). Since there is this composite definition, the field is non-simple or nonatomic and should be normalized by separating the data into simple relations. The end result of this example normalization would be three tables: one table PPN containing all those persons P who have a phone number, another table PUN containing all those persons P whose phone numbers are unknown, and lastly a table PNN containing all those persons P who have no phone number (Date 500). This can become extremely cumbersome as queries become longer and more complex, however it is easy to overcome the complexity by recreating the original table with a view which joins data from the decomposed/normalized tables (Date 505). Simply put, normalization results in a number of smaller atomic relations (Carpenter 382).

Now, there are various levels of database normalization of which only three or four are commonly applied: 1NF[1], 2NF, 3NF, and 4NF. Pertaining to levels of normalization above these, 5NF anomalies can typically be avoided with software routines and requires much more time than previous levels to achieve, 6NF is essentially only applicable for data warehousing which is not likely within the scope of initial database design, and DKNF describes is a theoretical perfect system which has never actually been accomplished (Carpenter 381-382). All databases begin at a trivial 0NF which is unnormalized (Carpenter 380). The first level of normalization, 1NF, must satisfy the requirement that all tuples (rows) in a table must be unique and attributes must be atomic (Ben-Gan 29). This means that no record should exist twice (since it would violate the definition of primary keys), and each attribute must be atomic, containing the same non array data type. The second level of normalization, 2NF, must satisfy 1NF and the requirement that nonkey attributes should not refer only to part of a candidate key (Ben-Gan 30). For example, any table with a multicolumn primary key such as customerid and storeid, all other attributes should pertain to the whole primary key and not just one of the columns (Ben-Gan 30). An attribute like location breaks 2NF because it can be found in the stores table without any reference to customers (Ben-Gan 30). The third level of normalization, 3NF, must satisfy 2NF and the requirement that all nonkey attributes must only depend on the primary key (Ben-Gan 31). All attributes only related to products should be on the products table, and all attributes only

[1] NF - Normal Form

related to stores should be on the stores tables, and all attributes that require both products and stores should be on their own table. For example, quantity is relative to both productid and storeid and therefore is on its own table. The fourth, and last that will be described, level of normalization, 4NF, must satisfy 3NF and the requirement that there cannot be any multivalued dependencies (Carpenter 379). For example, say a product table contained the attributes productid, year, and color. Since the year and color are independent of each other but are both dependent on productid, this table violates 4NF. After 4NF, the number of data anomalies are fewer and relations are more properly defined.

Businesses that employ database normalization should follow through to 4NF (Carpenter 382). The benefits of database normalization center around the nature of rightly defined data relations. First, normalization minimizes data redundancy by removing wasted space and repeated entries (Carpenter 380). This improves data integrity, reduces anomalies, and conserves storage space (Carpenter 380). Second, it strengthens the referential integrity of constraints based on the structure of the database (Carpenter 380). Third, data becomes easier to maintain and manipulate pertaining to data insertion, updates, and deletions (Carpenter 380). Lastly, normalization provides a more accurate database structure in terms of the problem it seeks to address and is able to easily adapt to future demands (Carpenter 380). On the other hand, while improving space consumption, database normalization reduces the efficiency of data retrieval and the ease of database management (Carpenter 382). As the various levels of normalization increase from 1NF to 4NF there is a significant return on investment and it is suggested to normalize through 4NF (Carpenter 381). Beyond this point however, most companies don’t need to continue normalization as it only pertains to special use cases (Carpenter 382).

Works Cited

Ben-Gan, Itzik. *T-SQL Fundamentals*. 3rd ed., Redmond, Microsoft Press, 2016.

Carpenter, D. A. (2008). Clarifying Normalization. *Journal of Information Systems Education*, *19*(4), 379–382. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=bsu&AN=36095235&site=eds-live&custid=s9004953

Date, Chris J. *SQL and Relational Theory: How to Write Accurate SQL Code*. 3rd ed., Sebastopol, O'Reilly Media Inc., 2015.