Database normalization

From Wikipedia, the free encyclopedia

Database normalization, or simply **normalization**, is the process of organizing the columns (attributes) and tables (relations) of a relational database to reduce data redundancy and improve data integrity.

Normalization involves arranging attributes in tables based on dependencies between attributes, ensuring that the dependencies are properly enforced by database integrity constraints. Normalization is accomplished through applying some formal rules either by a process of synthesis or decomposition. Synthesis creates a normalized database design based on a known set of dependencies. Decomposition takes an existing (insufficiently normalized) database design and improves it based on the known set of dependencies.

Edgar F. Codd, the inventor of the relational model (RM), introduced the concept of normalization and what we now know as the First normal form (1NF) in 1970.[1] Codd went on to define the Second normal form (2NF) and Third normal form (3NF) in 1971,[2] and Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974.[3] Informally, a relational database table is often described as "normalized" if it meets Third Normal Form.[4] Most 3NF tables are free of insertion, update, and deletion anomalies.

Contents

Objectives

A basic objective of the first normal form defined by Codd in 1970 was to permit data to be queried and manipulated using a "universal data sub-language" grounded in first-order logic.[5] (SQL is an example of such a data sub-language, albeit one that Codd regarded as seriously flawed.)[6]

The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

> 1. To free the collection of relations from undesirable insertion, update and deletion dependencies;
> 2. To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs;
> 3. To make the relational model more informative to users;

> 4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.
>
> — E.F. Codd, "Further Normalization of the Data Base Relational Model"[7]

The sections below give details of each of these objectives.

Free the database of modification anomalies

When an attempt is made to modify (update, insert into, or delete from) a table, undesired side-effects may arise in tables that have not been sufficiently normalized. An insufficiently normalized table might have one or more of the following characteristics:

- The same information can be expressed on multiple rows; therefore updates to the table may result in logical inconsistencies. For example, each record in an "Employees' Skills" table might contain an Employee ID, Employee Address, and Skill; thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each skill). If the update is not carried through successfully—if, that is, the employee's address is updated on some records but not others—then the table is left in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular employee's address is. This phenomenon is known as an **update anomaly**.
- There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" table might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code—thus we can record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly hired faculty member who has not yet been assigned to teach any courses except by setting the Course Code to null. This phenomenon is known as an **insertion anomaly**.
- Under certain circumstances, deletion of data representing certain facts necessitates deletion of data representing completely different facts. The "Faculty and Their Courses" table described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member, unless we set the Course Code to null in the record itself. This phenomenon is known as a **deletion anomaly**.

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

An **update anomaly**. Employee 519 is shown as having different addresses on different records.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201
424	Dr. Newsome	29-Mar-2007	?

An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

A **deletion anomaly**. All information about Dr. Giddens is lost if he temporarily ceases to be assigned to any courses.

Minimize redesign when extending the database structure

When a fully normalized database structure is extended to allow it to accommodate new types of data, the pre-existing aspects of the database structure can remain largely or entirely unchanged. As a result, applications interacting with the database are minimally affected.

Normalized tables, and the relationship between one normalized table and another, mirror real-world concepts and their interrelationships.

Example

Querying and manipulating the data within a data structure that is not normalized, such as the following non-1NF representation of customers, credit card transactions, involves more complexity than is really necessary:

Customer	Cust. ID	Transactions
Jones	1	Tr. ID / Date / Amount: 12890 / 14-Oct-2003 / −87; 12904 / 15-Oct-2003 / −50
Wilkins	2	Tr. ID / Date / Amount: 12898 / 14-Oct-2003 / −21
Stevens	3	Tr. ID / Date / Amount: 12907 / 15-Oct-2003 / −18; 14920 / 20-Nov-2003 / −70; 15003 / 27-Nov-2003 / −60

To each customer corresponds a *repeating group* of transactions. The automated evaluation of any query relating to customers' transactions therefore would broadly involve two stages:

1. Unpacking one or more customers' groups of transactions allowing the individual transactions in a group to be examined, and
2. Deriving a query result based on the results of the first stage

For example, in order to find out the monetary sum of all transactions that occurred in October 2003 for all customers, the system would have to know that it must first unpack the *Transactions* group of each customer, then sum the *Amounts* of all transactions thus obtained where the *Date* of the transaction falls in October 2003.

One of Codd's important insights was that this structural complexity could always be removed completely, leading to much greater power and flexibility in the way queries could be formulated (by users and applications) and evaluated (by the DBMS). The normalized equivalent of the structure above would look like this:

Customer	Cust. ID
Jones	1
Wilkins	2
Stevens	3

Cust. ID	Tr. ID	Date	Amount
1	12890	14-Oct-2003	−87
1	12904	15-Oct-2003	−50
2	12898	14-Oct-2003	−21
3	12907	15-Oct-2003	−18
3	14920	20-Nov-2003	−70
3	15003	27-Nov-2003	−60

In the modified structure, the keys are {Customer} and {Cust. ID} in the first table, {Cust. ID, Tr ID} in the second table.

Now each row represents an individual credit card transaction, and the DBMS can obtain the answer of interest, simply by finding all rows with a Date falling in October, and summing their Amounts. The data structure places all of the values on an equal footing, exposing each to the DBMS directly, so each can potentially participate directly in queries; whereas in the previous situation some values were embedded in lower-level structures that had to be handled specially. Accordingly, the normalized design lends itself to general-purpose query processing, whereas the unnormalized design does not. The normalized version also allows the user to change the customer name in one place and guards against errors that arise if the customer name is misspelled on some records.

List of Normal Forms

- UNF - "Unnormalized Form"
- 1NF - First Normal Form
- 2NF - Second Normal Form
- 3NF - Third Normal Form
- EKNF - Elementary Key Normal Form
- BCNF - Boyce–Codd Normal Form
- 4NF - Fourth Normal Form
- ETNF - Essential Tuple Normal Form (http://researcher.watson.ibm.com/researcher/files/us-fagin/icdt12.pdf)
- 5NF - Fifth Normal Form
- 6NF - Sixth Normal Form
- DKNF - Domain/Key Normal Form

See also

- Refactoring

Notes and references

1. Codd, E. F. (June 1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM*. **13** (6): 377–387. doi:10.1145/362384.362685.
2. Codd, E.F. "Further Normalization of the Data Base Relational Model". (Presented at Courant Computer Science Symposia Series 6, "Data Base Systems", New York City, May 24–25, 1971.) IBM Research Report RJ909 (August 31, 1971). Republished in Randall J. Rustin (ed.), *Data Base Systems: Courant Computer Science Symposia Series 6*. Prentice-Hall, 1972.

3. Codd, E. F. "Recent Investigations into Relational Data Base Systems". IBM Research Report RJ1385 (April 23, 1974). Republished in *Proc. 1974 Congress* (Stockholm, Sweden, 1974). , N.Y.: North-Holland (1974).
4. C.J. Date. *An Introduction to Database Systems*. Addison-Wesley (1999), p. 290
5. "The adoption of a relational model of data ... permits the development of a universal data sub-language based on an applied predicate calculus. A first-order predicate calculus suffices if the collection of relations is in first normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding (with appropriate syntactic modification) in a variety of host Ianguages (programming, command- or problem-oriented)." Codd, "A Relational Model of Data for Large Shared Data Banks" (http://www.acm.org/classics/nov95/toc.html), p. 381
6. Codd, E.F. Chapter 23, "Serious Flaws in SQL", in *The Relational Model for Database Management: Version 2*. Addison-Wesley (1990), pp. 371–389
7. Codd, E.F. "Further Normalization of the Data Base Relational Model", p. 34

Further reading

- Date, C. J. (1999), *An Introduction to Database Systems (http://www.aw-bc.com/catalog/academic/product/0,1144,0321197844,00.html)* (8th ed.). Addison-Wesley Longman. ISBN 0-321-19784-4.
- Kent, W. (1983) *A Simple Guide to Five Normal Forms in Relational Database Theory (http://www.bkent.net/Doc/simple5.htm)*, Communications of the ACM, vol. 26, pp. 120–125
- H.-J. Schek, P. Pistor Data Structures for an Integrated Data Base Management and Information Retrieval System

External links

- Database Normalization Basics (http://databases.about.com/od/specificproducts/a/normalization.htm) by Mike Chapple (About.com)
- Database Normalization Intro (http://www.databasejournal.com/sqletc/article.php/1428511), Part 2 (http://www.databasejournal.com/sqletc/article.php/26861_1474411_1)
- An Introduction to Database Normalization (http://mikehillyer.com/articles/an-introduction-to-database-normalization/) by Mike Hillyer.
- A tutorial on the first 3 normal forms (http://phlonx.com/resources/nf3/) by Fred Coulson
- DB Normalization Examples (http://www.dbnormalization.com/)
- Description of the database normalization basics (http://support.microsoft.com/kb/283878) by Microsoft
- Database Normalization and Design Techniques (http://www.barrywise.com/2008/01/database-normalization-and-design-techniques/) by Barry Wise, recommended reading for the Harvard MIS.
- A Simple Guide to Five Normal Forms in Relational Database Theory (http://www.bkent.net/Doc/simple5.htm)
- Normalization in DBMS by Chaitanya (beginnersbook.com) (http://beginnersbook.com/2015/05/normalization-in-dbms/)