Database normalization

From Wikipedia, the free encyclopedia

In the field of relational database design, normalization is a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics—insertion, update, and deletion anomalies—that could lead to a loss of data integrity.[1] E.F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the first normal form in 1970.[2] Codd went on to define the second and third normal forms in 1971,[3] and Codd and Raymond F. Boyce defined the Boyce-Codd normal form in 1974.[4] Higher normal forms were defined by other theorists in subsequent years, the most recent being the sixth normal form introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.[5]

Informally, a relational database table (the computerized representation of a relation) is often described as "normalized" if it is in the third normal form (3NF).[6] Most 3NF tables are free of insertion, update, and deletion anomalies, i.e. in most cases 3NF tables adhere to BCNF, 4NF, and 5NF (but typically not 6NF).

A standard piece of database design guidance is that the designer should create a fully normalized design; selective denormalization can subsequently be performed for performance reasons.[7] However, some modeling disciplines, such as the dimensional modeling approach to data warehouse design, explicitly recommend non-normalized designs, i.e. designs that in large part do not adhere to 3NF.[8]

Contents

- 1 Objectives of normalization
  - 1.1 Free the database of modification anomalies
  - 1.2 Minimize redesign when extending the database structure
  - 1.3 Make the data model more informative to users
  - 1.4 Avoid bias towards any particular pattern of querying
- 2 Background to normalization: definitions
- 3 Normal forms
- 4 Denormalization
  - 4.1 Non-first normal form (NF2 or N1NF)
- 5 Further reading
- 6 Notes and references
- 7 See also
- 8 External links

Objectives of normalization

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.[9] (SQL is an example of such a data sub-language, albeit one that Codd regarded as seriously flawed.)[10] Querying and manipulating the data within an unnormalized data structure, such as the following non-1NF representation of customers' credit card transactions, involves more complexity than is really necessary:
<table>
<thead>
<tr>
<th>Customer</th>
<th>Tr. ID</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>12890</td>
<td>14-Oct-2003</td>
<td>-87</td>
</tr>
<tr>
<td></td>
<td>12904</td>
<td>15-Oct-2003</td>
<td>-50</td>
</tr>
<tr>
<td>Wilkins</td>
<td>12898</td>
<td>14-Oct-2003</td>
<td>-21</td>
</tr>
<tr>
<td>Stevens</td>
<td>12907</td>
<td>15-Oct-2003</td>
<td>-18</td>
</tr>
<tr>
<td></td>
<td>14920</td>
<td>20-Nov-2003</td>
<td>-70</td>
</tr>
<tr>
<td></td>
<td>15003</td>
<td>27-Nov-2003</td>
<td>-60</td>
</tr>
</tbody>
</table>

To each customer there 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:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Tr. ID</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>12890</td>
<td>14-Oct-2003</td>
<td>-87</td>
</tr>
<tr>
<td>Jones</td>
<td>12904</td>
<td>15-Oct-2003</td>
<td>-50</td>
</tr>
<tr>
<td>Wilkins</td>
<td>12898</td>
<td>14-Oct-2003</td>
<td>-21</td>
</tr>
<tr>
<td>Stevens</td>
<td>12907</td>
<td>15-Oct-2003</td>
<td>-18</td>
</tr>
<tr>
<td>Stevens</td>
<td>14920</td>
<td>20-Nov-2003</td>
<td>-70</td>
</tr>
<tr>
<td>Stevens</td>
<td>15003</td>
<td>27-Nov-2003</td>
<td>-60</td>
</tr>
</tbody>
</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. All of the values in the data structure are on an equal footing: they are all exposed to the DBMS directly, and can directly participate 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 objectives of normalization beyond 1NF were stated as follows by Codd:
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 follow. Not all tables can suffer from these side-effects; rather, the side-effects can only 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 of his skills). 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. This phenomenon is known as an **insertion anomaly**.

- There are circumstances in which the deletion of data representing certain facts necessitates the 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. This phenomenon is known as a deletion anomaly.

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.

Make the data model more informative to users

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

Avoid bias towards any particular pattern of querying

Normalized tables are suitable for general-purpose querying. This means any queries against these tables, including future queries whose details cannot be anticipated, are supported. In contrast, tables that are not normalized lend themselves to some types of queries, but not others.

Background to normalization: definitions

- **Functional dependency**: Attribute B has a functional dependency on attribute A (i.e., A \( \rightarrow \) B) if, for each value of attribute A, there is exactly one value of attribute B. If value of A is repeating in tuples then value of B will also repeat. In our example, Employee Address has a functional dependency on Employee ID, because a particular Employee ID value corresponds to one and only one Employee Address value. (Note that the reverse need not be true: several employees could live at the same address and therefore one Employee Address value could correspond to more than one Employee ID. Employee ID is therefore not functionally dependent on Employee Address.) An attribute may be functionally dependent either on a single attribute or on a combination of attributes. It is not possible to determine the extent to which a design is normalized without understanding what functional dependencies apply to the attributes within its tables; understanding this, in turn, requires knowledge of the problem domain. For example, an Employer may require certain employees to split their time between two locations, such as New York City and London, and therefore want to allow Employees to have more than one Employee Address. In this case, Employee Address would no longer be functionally dependent on Employee ID.

Another way to look at the above is by reviewing basic mathematical functions:

Let \( F(x) \) be a mathematical function of one independent variable. The independent variable is analogous to the attribute A. The dependent variable (or the dependent attribute using the terminology above), and hence the term functional dependency, is the value of \( F(A) \); A is an independent attribute. As we know, mathematical functions can have only one output. Notationally speaking, it is common to express this relationship in mathematics as \( F(A) = B \), or \( B \rightarrow F(A) \).

There are also functions of more than one independent variable—commonly, this is referred to as multivariable functions. This idea represents an attribute being functionally dependent on a combination of attributes. Hence, \( F(x,y,z) \) contains three independent variables, or independent attributes, and one