Description of the database normalization basics

Article • 07/13/2023 • Applies to: Access 2021, Access 2019, Access 2016, Access 2013

This article explains database normalization terminology for beginners. A basic understanding of this terminology is helpful when discussing the design of a relational database.

Description of normalization

Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is easier to implement if that data is stored only in the Customers table and nowhere else in the database.

What is an "inconsistent dependency"? While it's intuitive for a user to look in the Customers table for the address of a particular customer, it may not make sense to look there for the salary of the employee who calls on that customer. The employee's salary is related to, or dependent on, the employee and thus should be moved to the Employees table. Inconsistent dependencies can make data difficult to access because the path to find the data may be missing or broken.

There are a few rules for database normalization. Each rule is called a "normal form." If the first rule is observed, the database is said to be in "first normal form." If the first three rules are observed, the database is considered to be in "third normal form." Although other levels of normalization are possible, third normal form is considered the highest level necessary for most applications.

As with many formal rules and specifications, real world scenarios don't always allow for perfect compliance. In general, normalization requires additional tables and some customers find this cumbersome. If you decide to violate one of the first three rules of normalization, make sure that your application anticipates any problems that could occur, such as redundant data and inconsistent dependencies.

The following descriptions include examples.

First normal form

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key.

Don't use multiple fields in a single table to store similar data. For example, to track an inventory item that may come from two possible sources, an inventory record may contain fields for Vendor Code 1 and Vendor Code 2.

What happens when you add a third vendor? Adding a field isn't the answer; it requires program and table modifications and doesn't smoothly accommodate a dynamic number of vendors. Instead, place all vendor information in a separate table called Vendors, then link inventory to vendors with an item number key, or vendors to inventory with a vendor code key.

Second normal form

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

Records shouldn't depend on anything other than a table's primary key (a compound key, if necessary). For example, consider a customer's address in an accounting system. The address is needed by the Customers table, but also by the Orders, Shipping, Invoices, Accounts Receivable, and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables, store it in one place, either in the Customers table or in a separate Addresses table.

Third normal form

- Eliminate fields that don't depend on the key.

Values in a record that aren't part of that record's key don't belong in the table. In general, anytime the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

EXCEPTION: Adhering to the third normal form, while theoretically desirable, isn't always practical. If you have a Customers table and you want to eliminate all possible interfield dependencies, you must create separate tables for cities, ZIP codes, sales representatives, customer classes, and any other factor that may be duplicated in multiple records. In theory, normalization is worth pursuing. However, many small tables may degrade performance or exceed open file and memory capacities.

It may be more feasible to apply third normal form only to data that changes frequently. If some dependent fields remain, design your application to require the user to verify all related fields when any one is changed.

Other normalization forms

Fourth normal form, also called Boyce-Codd Normal Form (BCNF), and fifth normal form do exist, but are rarely considered in practical design. Disregarding these rules may result in less than perfect database design, but shouldn't affect functionality.

Normalizing an example table

These steps demonstrate the process of normalizing a fictitious student table.

1. Unnormalized table:

Expand table

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	101-07	143-01	179-04

2. First normal form: No repeating groups

Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, and Class3 in the above records are indications of design trouble.

Spreadsheets often use the third dimension, but tables shouldn't. Another way to look at this problem is with a one-to-many relationship, don't put the one side and the many sides in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class#), as shown in the following example:

Expand table

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

3. Second normal form: Eliminate redundant data

Note the multiple **Class#** values for each **Student#** value in the above table. Class# isn't functionally dependent on Student# (primary key), so this relationship isn't in second normal form.

The following tables demonstrate second normal form:

Students:

Expand table

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration:

Expand table

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

4. Third normal form: Eliminate data not dependent on key

In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute. The solution is to move that attribute from the Students table to the Faculty table, as shown below:

Students:

Expand table

Student#	Advisor
1022	Jones
4123	Smith

Faculty:

Expand table

Name	Room	Dept
Jones	412	42
Smith	216	42

Feedback

Was this page helpful? Yes No

Provide product feedback