Title

SQL and Database Design for web development - Chapter I.

About the Author

Sérgio Fontes it's a Software & Database Analyst engineer. With academic background in Maths and Informatics, Sérgio worked on several projects under the EC's IST (Information Society Technologies) programme as a programmer and also as project manager of the Portuguese team.

Abstract

Database normalization is what all database designers seek. However the database design for websites it's very complex and specific, according with specifications of the data to be stored it's more acceptable, to allow data redundancy in order to reduce, the time of execution on an sql query and the number of tables present on the statement.

Keywords

SQL; TSQL; Database; ER; Redundancy; Normalization; Websites; Data; Web;

About this Article

In this series of articles I want to analyze and describe database and query development for web solutions, as development technology we will use SQL and Microsoft SQL Server 2000 as the database engine. On this first article I want to give special emphasis on the Database design and what can happen if this is not done properly, let's see why:

Content

Database and Query development for web solutions such as dynamic websites and web based applications, is a complex and difficult task. On this type of solutions band with, speed and server response are key points that can slow down you software if your queries aren't good enough. However you may be an excellent sql query builder but in the end it also depends of your database design.

Consider that we want to store product and product quality data on a database to be available on a website. According with the rules of normalization we would have the following ER[1]:

[1] ER it's the Entity Relationship diagram.

Picture 1: Entity Relationship Diagram.

If you don't have knowledge of ER or forgot it, keep in mind that you want to organize your data in the proper way to avoid data redundancy.

Now let's build the query to publish product data on a webpage. The SQL query would be the following:

*SELECT * FROM products INNER JOIN quality ON products.prod_quality=quality._quality_id*

or

SELECT prod_id,prod_name, (SELECT quality_name WHERE quality_id=products.prod_quality) AS prodquality FROM products

Any of the above mentioned sql statement needs to access to 2 tables and not only one. This means that if these tables store a large number of items the database engine would spend considerable time.

The Normalization it's a difficult task and many time is spent by database developers. In my opinion we cannot use a blind rule, we must work on a case by case analysis. In the end it all depends of the specificity of the data you want to store and sometimes if possible and needed we should have some redundancy.

Let's consider the following scenario:

Each product will be classified as excellent, good, normal or bad. In my opinion we can make same changes to the ER present on picture 1 that will allow us to speed up the query time and reduce the number of tables present on the statement. I would definitely change the ER to the following:

Picture 2: New Entity Relationship Diagram.

But know I have data duplication!

That's correct but it's a more suitable option because if I want publish product data on a webpage. The SQL query would be the following:

*SELECT * FROM products*

Witch is a considerable improvement from the previous sql statement, wouldn't you say?

The number of tables to "visit" was reduced in 50% and the time of execution as also been reduced.

The above example uses a "*one to many"* relationship but you can also use it on "*many to many"* relationships.

Another common example, where the careful use of data redundancy is in my opinion imperative is on, invoice header and invoice detail tables. But in this situation we can find two different purposes:

1. The fist one, similar to the initial example, is to copy client information present in the header of the invoice to the Invoice Header table. By doing this when you need to publish the invoice on the web you will only need to query 2 tables instead of 3. This means that you will only make one inner join instruction (between Invoice Header and Invoice Details tables) and not 2 inner join instructions (between Invoice Header and invoice details tables and between Invoice Header and Client tables).

2. Clients can change there delivery address. If you use the normalization rule blindly you would do the following:

   Invoice Header table as a foreign key that connects this table with the Client table. If you need to publish the invoice header information you simply use an *inner join* statement and retrieve the client address.

   Now let's imagine that we have a set of invoices and then the client updates is address info. If I want to publish the invoice header of an invoice created before the address update, I will use the mentioned *inner join* statement, but this statement gets the info that is currently on the client table that was updated. OH MY, DATA AS BEEN LOST!

As you can see each case it's a unique case and once again we need to analyze in details the data that we need to store.

Conclusion

As you can see in particular cases we can win on query speed by allowing data redundancy. But pay attention this type of solution, depends of table structures and in many cases this can't be done. For example, if the fields of the foreign table needed more then one item then this solution couldn't be used and we would need to use the first sql statement.

If needed, but only if needed, use data redundancy, this will allow you to increase query speed and also to prevent you from loosing data.

Contact Info

Sérgio Fontes
Sergio@computencial.com