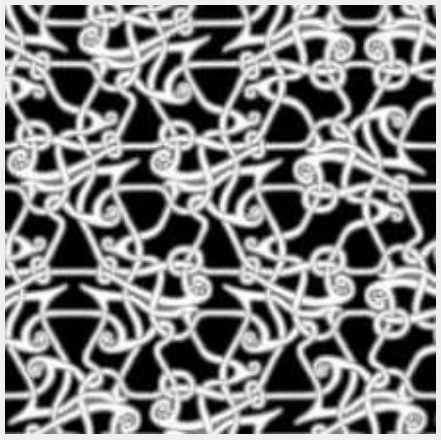
Database Normalization Explained in Simple English

Introduction



Database normalization is process used to organize a database into tables and columns. The idea is that a table should be about a *specific* topic and that only those columns which support that topic are included. For example, a spreadsheet containing information about sales people and customers serves several purposes:

- Identify sales people in your organization
- List all customers your company calls upon to sell product
- Identify which sales people call on specific customers.

By limiting a table to one purpose you reduce the number of duplicate data that is contained within your database, which helps eliminate some issues stemming from database modifications. To assist in achieving these objectives, some rules for database table organization have been developed. The stages of organization are called normal forms; there are three normal forms most databases adhere to using. As tables satisfy each successive normalization form, they become less prone to database modification anomalies and more focused toward a sole purpose or topic. Before we move on be sure you understand the definition of a database table.

Reasons for Normalization

There are three main reasons to normalize a database. The first is to minimize duplicate data, the second is to minimize or avoid data modification issues, and the third is to simplify queries. As we go through the various states of normalization we'll discuss how each form addresses these issues, but to start, let's look at some data which hasn't been normalized and discuss some potential pitfalls. Once these are understood, I think you'll better appreciate the reason to normalize the data. Consider the following table:

SalesStaff						
EmployeeID	**SalesPerson**	**SalesOffice**	**OfficeNumber**	**Customer1**	**Customer2**	**Customer3**
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		

Note: The primary key columns are underlined

The first thing to notice is this table serves many purposes including:

1. Identifying the organization's salespeople
2. Listing the sales offices and phone numbers
3. Associating a salesperson with an sales office
4. Showing each salesperson's customers

As a DBA this raises a red flag. In general I like to see tables that have one purpose. Having the table serve many purposes introduces many of the challenges; namely, data duplication, data update issues, and increased effort to query data.

Data Duplication and Modification Anomalies

Notice that for each SalesPerson we have listed both the SalesOffice and OfficeNumber. This information is duplicated for each SalesPerson. Duplicated information presents two problems:

1. It increases storage and decrease performance.

2. It becomes more difficult to maintain data changes.

For example

- Consider if we move the Chicago office to Evanston, IL. To properly reflect this in our table, we need to update the entries for all the SalesPersons currently in Chicago. Our table is a small example, but you can see if it were larger, that potentially this could involve hundreds of updates.

- Also consider what would happen if John Hunt quits. If we remove his entry, then we lose the information for New York.

These situations are modification anomalies. There are three modification anomalies that can occur:

Insert Anomaly

There are facts we cannot record until we know information for the entire row. In our example we cannot record a new sales office until we also know the sales person. Why? Because in order to create the record, we need provide a primary key. In our case this is the

EmployeeID.

Update Anomaly

The same information is recorded in multiple rows. For instance if the office number changes, then there are multiple updates that need to be made. If these updates are not

successfully completed across all rows, then an inconsistency occurs.

Deletion Anomaly

Deletion of a row can cause more than one set of facts to be removed. For instance, if John Hunt retires, then deleting that row cause use to lose information about the New York office.

Search and Sort Issues

The last reason we'll consider is making it easier to search and sort your data. In the SalesStaff table if you want to search for a specific customer such as Ford, you would have to write a query like

```
SELECT SalesOffice
FROM SalesStaff
WHERE Customer1 = 'Ford' OR
      Customer2 = 'Ford' OR
      Customer3 = 'Ford'
```

Clearly if the customer were somehow in one column our query would be simpler. Also, consider if you want to run a query and sort by customer. The way the table is currently defined, this isn't possible, unless you use three separate queries with a UNION. These anomalies can be eliminated or reduced by properly separating the data into different tables, to house the data in tables which serve a single purpose. The process to do this is called normalization, and the various stages you can achieve are called the normal forms.

Definition of Normalization

There are three common forms of normalization: 1st, 2nd, and 3rd normal form. There are several additional forms, such as BCNF, but I consider those advanced, and not too necessary to learn in the beginning. The forms are progressive, meaning that to qualify for 3rd normal form a table must first satisfy the rules for 2nd normal form, and 2nd normal form must adhere to those for 1st normal form. Before we discuss the various forms and rules in details, let's summarize the various forms:

- **First Normal Form** – The information is stored in a relational table and each column contains atomic values, and there are not repeating groups of columns.
- **Second Normal Form** – The table is in first normal form and all the columns depend on the table's primary key.
- **Third Normal Form** – the table is in second normal form and all of its columns are not transitively dependent on the primary key

Do not get too hung up if you don't know what these rules mean at the moment; we'll explain them in detail in the next post using examples. For now it's important to understand there are three rules which build upon each other. More tutorials are to follow! Remember! I want to remind you all that if you have other questions you want answered, then post a comment or tweet me. I'm here to help you. What other topics would you like to know more about?

Kris Wenzel | Categories ↓ | Tags ↓

Here's some more SQL articles we think you'll like!

Database Third Normal Form Explained in Simple English

Database Second Normal Form Explained in Simple English

Database Indexes Explained

What is a Non-Equi Join in SQL and What is its Purpose?

How can I find Duplicate Values in SQL Server?

BI For Beginners – a Business Intelligence Interview with Brian Larson

Learn about the First Normal Form and Database Design

Build Dynamic SQL in a Stored Procedure

Kris Wenzel

Kris Wenzel has been working with databases over the past 28 years as a developer, analyst, and DBA. He has a BSE in Computer Engineering from the University of Michigan and an MBA from the University of Notre Dame. Kris has written hundreds of blog articles and many online courses. He is loves helping others learn SQL.

Click Here to Leave a Comment Below 37 comments

Chris Null

Question on normalization in SQL relational dbs.

Let's say I have an application where a given person can be included as a customer, a employee and/or a referrer .

In the customer table you have last_name, first_name, Middle_name, dob, gender and staff id as pk identity.
fk employee_id and fk referrer_id

employee table has name, dob, gender, employee_id as pk identity

referrer table has name, and referrer_id as pk identity

It seems like it might be better to use the person table with a column for employee_id and referrer_id that link to a table that doesn't have repetitive information in it like names or DOB so that when you query the data you don't get a weird name or somehow conflicting data. Duplication isn't covered under normalization but it seems like a bad idea to have duplicated data in a relational database. What is the rule/practice for removing duplicated data from the db?

Reply

Kris Wenzel

Hi Chris,
Your hunch is correct. It would be better to have a person table to house customers, employees, and referrers.
In this case normalization doesn't directly speak towards having duplicate data in different tables. I think you could consider the table normalized, but of course it isn't the design we're looking for.
We can look to generalization and specialization concepts to guide our design. Specifically, the common attributes can be place in a generalized entity, which in our case, is a person.
The specialized attributes, are then placed in their own entities: customer, employee, and referrer.

This is how I would lay out the tables.

```
person: (personID [pk], last_name, first_name, middle_name, dob, gender)
```

```
customer: (customerID [pk], last_purchase_date)
• FK: customerID to person.personID

employee: (employeeID [pk], date_hired)
• FK: employeeID to person.personID

referrer: (referrerID [pk], referral_date)
• FK: referrerID to person.personID
```

Notes:
[pk] = primary key.

Reply

Vadim Tropashko

Database normalization is fascinating topic, although somewhat disconnected from query. A typical introductory database course teaches students what Relational Algebra and SQL are, and then switches gears and talks about normalization with no leverage of skills developed at the first half of the course!

Speaking of design with Customer1, Customer2, Customer3 attributes: http://www.orafaq.com/usenet/comp.databases.theory/2001/05/17/0244.htm

Reply

Kris Wenzel

Hi Vadmin,

Thanks for your comments and I'm glad you are finding my topics fascinating.

My target audience are people that are just starting to learn SQL. I'm trying to get across important principles using simple and

easy to understand English. I'm trying to keep it non-technical, which is hard when you are dealing with a technical subject!

Regarding normalization, the reason I did it before exploring joins is that I thought it would be good to give my readers an appreciation of why their data is spread across so many tables and why they have to go through the hassle to learn how to stitch the data back together.

Reply

Polo

Note, please note, that a slight denormalisation **CAN** be a freaking speedup for your system...
Lookup-tables tend to be logical-read hogs, even when they're small. You'll just need to lookup **a lot**!

So your reason number 1 for "Data Duplication and Modification Anomalies" is not a rule written in marble.

P.

Reply

Kris Wenzel

Hi,

I agree that the rule shouldn't be written in marble (good metaphor btw).

There are definitely times when I take a step back and don't follow strict normalization. I've done this many times in the applications I've written over the years; however, I think in general normalization does help performance more than it hinders it.

Check out the end of my post on the third normal form. I talk about cases when normalization can get out of hand.

Reply

YWong

Where's Get Ready to Learn SQL. 7? Thanks

Reply

Kris Wenzel

Hi, You can find the 7th article here!

Reply

vishnu

I want you to explain concurrency control topic in simple words

Reply

Kris Wenzel

Hi Vishnu, I'll add concurrency to my list of topics. It is a good one to cover. Be sure to subscribe to my email so you'll know when it's published.

Reply

dennis morgan

Nicely done. As clear and succinct as one could ask for.

Reply

Kris Wenzel

Thanks Dennis. I appreciate the compliment. If there are other DB topics you think are confusing, and you would like me to write about, please let me know. You can email me from my Contact page (above on the menu) or leave a reply here.

Thanks!

Reply

Newton Makeni

Thanks for the job well done Kris, I have read several blogs and Tech-journals none has been this clear and simplified. Thanks again.

Reply

Kris Wenzel

Thanks for letting me know. I really appreciate that you like the articles. If there are other SQL topics you would like me to explore or explain, please let me know.

Reply

Vuroborox

Hi Kris,

Great article series! It came in handy for a student like myself.

Reply

Kris Wenzel

Hi – I'm glad you liked the series. :)

Reply

sachin

hi
the post is very good and it is very easy to understand thanks for it
if u have tutorials about jsf using visual web ice faces can u plzz send it

Reply

Kris Wenzel

Hi,
I'm sorry but I don't have anything about jsf. I do know tons of C#, so if you're interested in seeing how c# and SQL can go together let me know.

I was a developer for many many years (since 1987). Luckily I didn't have to learn COBOL...

Reply

Leave a Reply:

Name* Email* Website

SUBMIT

←Previous post

Next post→