# The Making of My Database

**Jeremy Whitenect**

2023-12-12

—

Database Fundamentals

—

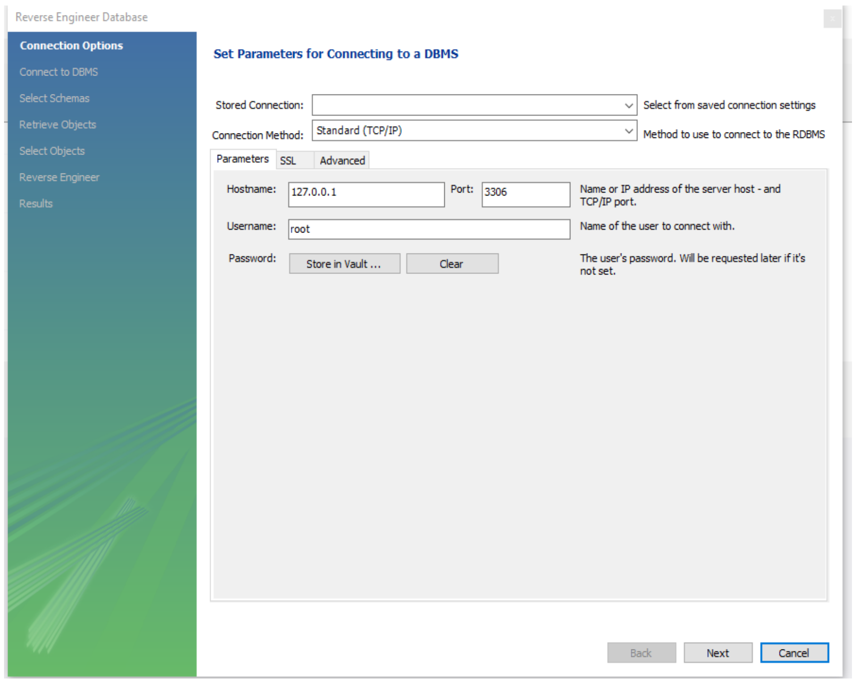
Nadia Gouda

## Table of Contents

## General Structure

Although clear in instruction, the structure of the database sadly does not implement itself, that is my job. The first step is an easy one, simply to add the four tables we have been instructed to make into a schema. This could be done using the SQL Online Compiler we have been using throughout this course, but I found it quite cumbersome for something on this scale. I decided to go with MySQL due to its user-friendly interface and popularity, thus having many tutorials should something go awry (they did indeed go awry, perhaps not in the way you may think however).  Still, things went quite smoothly in terms of setting up the program, everything was clear, but I did use a tutorial by the Youtuber Programming with Mosh just in case.

After exploring the program, myself for a little while to better my understanding of it, I found out that one could use a graphical user interface method to create tables, insert data and probably do many other things. But, because of the nature of this project and this class as whole I made a personal decision to not use that method, instead strictly using the standard SQL statements we have been taught. And, although perhaps not something that is completely related to the actual statements themselves, I did use the "reverse engineer" menu (see Figure 1.1) to create the entity relationship diagram that will be pictured later. After I had finished my exploration of the program, it was time for me to finally create the tables of my schema that I had aptly titled "book_store." Although my .sql files are included with this document, I have still pictured the SQL code (see Figure 1.2) to add to the ease of readability. As you can see, there is nothing too out of the ordinary, although one thing may jump out at you if you have a keen eye. I decided to have all my actual images for this document be in the program Notepad++ as I feel it gives a clearer picture of what's going on since it color codes commands in a better way. We have all of our fields as well as the primary key, the only thing being truly odd about the SQL is the name field which we shall get to later. But, other than that, it's the same process with only four tables. Although, I added some of the foreign keys to the tables that required them in a separate file due to forgetfulness.



Set Parameters for Connecting to a DBMS

Stored Connection: Select from saved connection settings

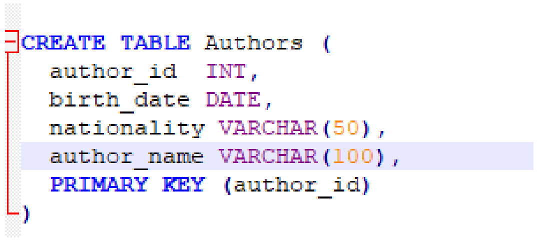
Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Hostname: 127.0.0.1 Port: 3306 Name or IP address of the server host - and TCP/IP port.

Username: root Name of the user to connect with.

Password: The user's password. Will be requested later if it's not set.

*Figure 1.1: Image of the reverse engineer menu in the program MySQL.*

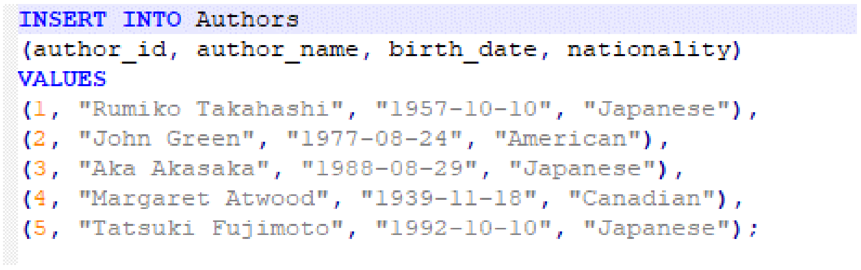


```
CREATE TABLE Authors (
  author_id  INT,
  birth_date DATE,
  nationality VARCHAR(50),
  author_name VARCHAR(100),
  PRIMARY KEY (author_id)
)
```

*Figure 1.2: Image of SQL code to create a table.*

## Data Insertion

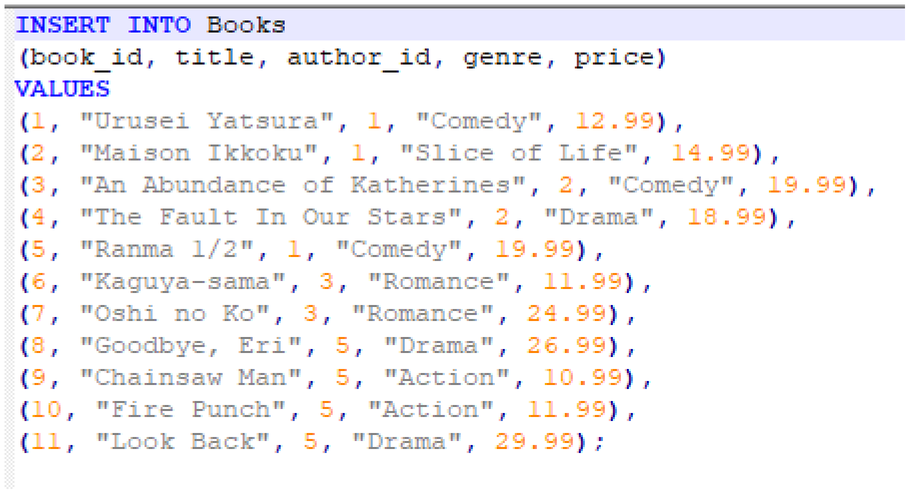
The next step in both this project and in general when it comes to creation of tables in a schema is the insertion of data. There was a fair amount of data to be inserted as 5 records were required per table and I decided to add even more for some tables as it made for easier querying when it came to that section of the project because of some of the required queries.  Although I could have inserted every piece of data in one .sql file, I opted to have the data be inserted for each table in it's own file for clarity. Once again, I will use the Authors table as an example (see Figure 2.1).



```
INSERT INTO Authors
(author_id, author_name, birth_date, nationality)
VALUES
(1, "Rumiko Takahashi", "1957-10-10", "Japanese"),
(2, "John Green", "1977-08-24", "American"),
(3, "Aka Akasaka", "1988-08-29", "Japanese"),
(4, "Margaret Atwood", "1939-11-18", "Canadian"),
(5, "Tatsuki Fujimoto", "1992-10-10", "Japanese");
```

*Figure 3.1: Image of SQL code to insert data into a table.*

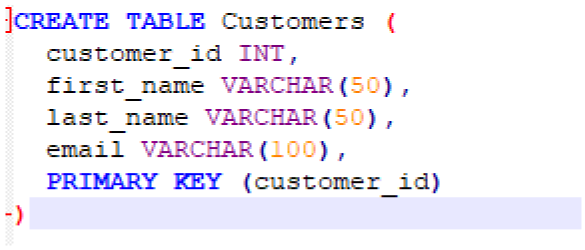
Of course, since this is meant to be a project and not a real database there is no sensitive information here and the author_id field has it's values be consecutively numbered when this would not be the case in a real database as this is bad practice.  As this is the same table used in my previous example, it too has the same issue with the name field which is now even more glaring since you can now clearly see that there are two pieces of data in the field. Another thing about this field is that all the names are formatted in the same way, first name followed by a space and then followed by the last name. This will be an important piece of information used later.  But, once again, nothing here seems out of the ordinary. All the data is inserted as it should be and since this is one of the basic elements of SQL, I didn't encounter any issues. As I personally have some trouble with attempting to find small details, and thus, errors in code, this is something I was very happy to get correct on my first try, even taking into account the simplicity of it all. As an example of another group of inserted values with no jarring issues, see Figure 2.2. As you can see, I have decided to go with real books and authors as, after working on another project, most of these were fresh in my mind. I haven't much more to say as this is one of the most basic things in SQL, so this should suffice.



```
INSERT INTO Books
(book_id, title, author_id, genre, price)
VALUES
(1, "Urusei Yatsura", 1, "Comedy", 12.99),
(2, "Maison Ikkoku", 1, "Slice of Life", 14.99),
(3, "An Abundance of Katherines", 2, "Comedy", 19.99),
(4, "The Fault In Our Stars", 2, "Drama", 18.99),
(5, "Ranma 1/2", 1, "Comedy", 19.99),
(6, "Kaguya-sama", 3, "Romance", 11.99),
(7, "Oshi no Ko", 3, "Romance", 24.99),
(8, "Goodbye, Eri", 5, "Drama", 26.99),
(9, "Chainsaw Man", 5, "Action", 10.99),
(10, "Fire Punch", 5, "Action", 11.99),
(11, "Look Back", 5, "Drama", 29.99);
```

*Figure 2.2: Image of SQL code to insert data into a table.*

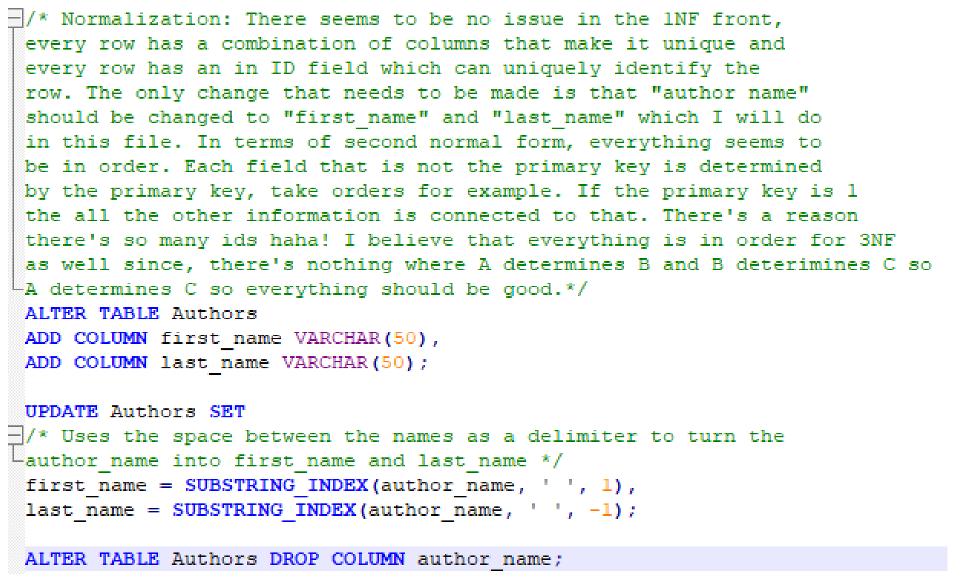
## Normalization



```
CREATE TABLE Customers (
   customer_id INT,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   email VARCHAR(100),
   PRIMARY KEY (customer_id)
)
```

*Figure 3.1: Image of SQL code to make a table.*

Much to my surprise (and to my delight) there was only one problem with the structure of the schema that we were instructed to create. The field, name, in the Authors table should have been two fields, fir st_name and last_name just like in the Customers table (see figure 3.1). There are, of course, many ways to do this. A potentially tedious but still viable way to do so would be to create the fields for the first name and the last name, then manually insert the data from the name field, put the respective names into their respective field and then finally delete the original name field. In a poorly created database or if there were some forms of anomaly, this could be a requirement rather than a tedious option. Either way, I did things differently and in quite an interesting way if you ask me. As you can see in Figure 3.2, there are three SQL statements (I suppose more than three depending on how you see things but I'm basing it on how many semicolons can be used). The first is nothing new, simply adding the two required fields so that there is only one piece of information pertaining to each field. The third statement isn't new but is still similar in its simplicity, doing nothing more than deleting the newly irrelevant field after everything is all said and done. Now, for the exceedingly interesting second SQL statement. This uses SUBSTRING_INDEX and something we used early on in this course but not with SQL, a delimiter. SUBSTRING_INDEX first takes the field from which it will spilt into two, then the quotes that seem empty aren't! When inserting the data into the names field, I made sure they all followed the same format for consistently yes, but also because I was almost sure there would be a way to use a delimiter in SQL for something like this. A bit of searching on W3Schools later and, sure enough, I had my answer. Because of my clever thinking, I was able to move this data into it's rightful place, using the space between the names as a delimiter and the newly found function. Thankfully, I was able to do this part quite swiftly. Also, since I have yet to mention, the positive and negative 1's, it is the number of times to search for the delimiter, positive numbers search to the left of the delimiter and negative to the right. I know it's not best to get a big head, but I was quite pleased with how this section of the project turned out, this part certainly got me to learn even more SQL.

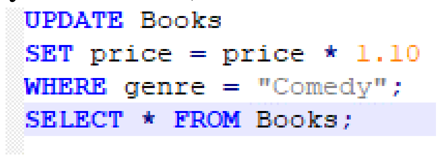


```
/* Normalization: There seems to be no issue in the 1NF front,
every row has a combination of columns that make it unique and
every row has an in ID field which can uniquely identify the
row. The only change that needs to be made is that "author name"
should be changed to "first_name" and "last_name" which I will do
in this file. In terms of second normal form, everything seems to
be in order. Each field that is not the primary key is determined
by the primary key, take orders for example. If the primary key is 1
the all the other information is connected to that. There's a reason
there's so many ids haha! I believe that everything is in order for 3NF
as well since, there's nothing where A determines B and B deterimines C so
A determines C so everything should be good.*/
ALTER TABLE Authors
ADD COLUMN first_name VARCHAR(50),
ADD COLUMN last_name VARCHAR(50);

UPDATE Authors SET
/* Uses the space between the names as a delimiter to turn the
author_name into first_name and last_name */
first_name = SUBSTRING_INDEX(author_name, ' ', 1),
last_name = SUBSTRING_INDEX(author_name, ' ', -1);

ALTER TABLE Authors DROP COLUMN author_name;
```

*Figure 3.2: Image of SQL code to edit a table to comply with first normal form.*

## SQL Queries

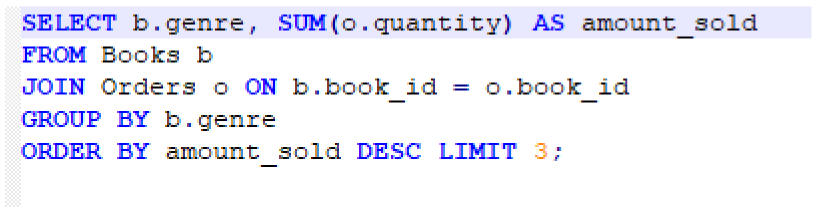
We were tasked with creating 8 SQL queries although this section will focus on the on the non-advance queries. If you take a look at Figure 4.1, you will see one the one of the 5 queries we were required to create for the simpler queries section. This is one of the more interesting yet still simple queries where we were tasked with increasing the price of a specific genre by 10%. Doing this specific query was very reminiscent of Java to me. Setting the price to itself and then multiplying it by 1.10 was something that jumped out to me as very Javaesque. Then all I had to do was add a WHERE (or an if if we're still using the Java analogy) to make it so that only one genre has it's price increased. In this case, Comedy, those chuckles don't come cheap. Lastly, I just displayed the table, sometimes I did this in the query itself and, since all I wanted to do was check if everything worked as intended, sometimes I would open another SQL file and quickly test without saving since all it would have would be something along the lines of "SELECT * FROM Books;" This was a very simple query all things considered.



```sql
UPDATE Books
SET price = price * 1.10
WHERE genre = "Comedy";
SELECT * FROM Books;
```

*Figure 4.1: Image of SQL code to increase the price of a specific genre.*

The next query I did (see Figure 4.2) was a bit more complicated and that's where I encountered my first big issue. And what's funny is, the issue was not with the actual query itself but with my lack of notice with something. It's not shown here but I had o.book_id as o.order_id by mistake. As this was more of a semantic error rather than a full-on error, the code still ran. At first, I didn't notice anything odd, but as it is good practice, I still compared the result of this query to the data itself and found out that the amount sold did not actually match with genre. As I mentioned earlier, I don't have the easiest time weeding out small details, combined with the slightly more complex nature of this query, I thought I simply had the query wrong. I changed it and tried a few other things while keeping the original query intact as a copy just in case. Luckily, I did because after quite a bit of frustration I happened to look back at it and noticed the, what should have been, glaring issue. But what matters is that I still managed to find it and the query worked perfectly afterwords. And, I suppose this did have some degree of enjoyment to it, when I realized what the problem was and fixed it and was practically jumping with joy when everything worked as it should! The other queries aren't nearly as interesting so it would be quite annoyingly repetitive if I were to go over the other three queries for this section. As such, I think it best that I move on to the section on the more advanced queries.

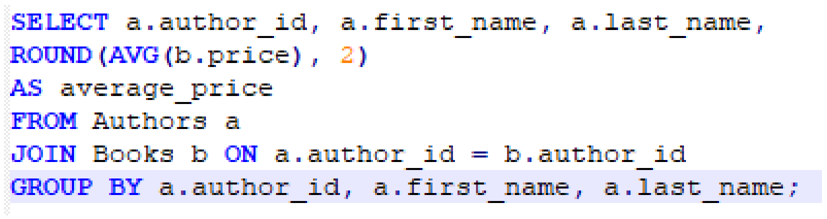


```sql
SELECT b.genre, SUM(o.quantity) AS amount_sold
FROM Books b
JOIN Orders o ON b.book_id = o.book_id
GROUP BY b.genre
ORDER BY amount_sold DESC LIMIT 3;
```

*Figure 4.2: Image of SQL code to get the top three best selling genres.*

## Advanced SQL Queries

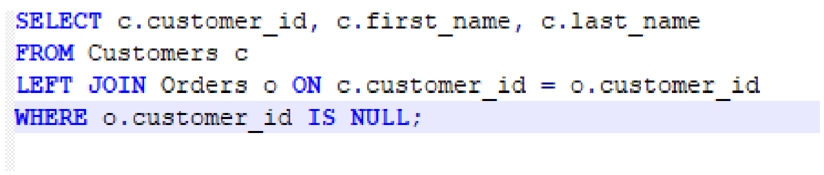
Built in SQL methods, my favorite! For this, we were instructed to get the average price of each author's overall published books (see Figure 5.1). So, I got the author_id and both names for presentation purposes and then used the AVG method to get the average and the ROUND method to present the averages as two decimal points which makes sense since we're working with money here. Next, I use an alias to make it clear what is being shown in the query followed by a join and a group by, so I get the result that I intended. This one was a bit of a thinker to be honest, it wasn't so much that I didn't understand what to do it was just that I had to put all the puzzle pieces in the correct places. Still though, I'm quite fond of puzzles and so the ones provided by SQL are no exception. This one was probably the one the hardest of the three provided we do in the advanced section of the SQL queries. In fact, checking which customers have no orders (see Figure 5.2) and checking which authors have published no books (see Figure 5.3) were much easier (to me at least)! First, they're the same query. Well, not in a literal sense but they're asking the same question with different numbers so to speak. Honestly, I would have done things slightly differently if I were to make the advanced queries section.



```
SELECT a.author_id, a.first_name, a.last_name,
ROUND(AVG(b.price), 2)
AS average_price
FROM Authors a
JOIN Books b ON a.author_id = b.author_id
GROUP BY a.author_id, a.first_name, a.last_name;
```

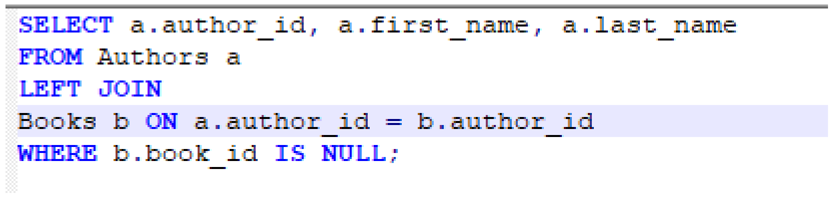
*Figure 5.1: Image of SQL code to get the average price of an author's books.*

I would first, remove one of these queries, as I said, they might as well be the exact same thing. Then, I would swap this now singular query with the three best selling genres query from the previous section. Even bearing my personal difficulties with it, I do believe it to be a bit more difficult in general than these two. Then, so the numbers all add up, I would create a new query for this advanced section. That's just what would do, maybe, for some reason, that would meet the outcomes of this course, I honestly don't know. That's just what I would do and, even if it cannot be acted upon, I still think it's important to give one's input. Still though, that's all hypothetical and it's not as if these queries were bad by any means, maybe just a bit confusing in how their ordered. But this is just my perspective, for all I know, I'm the only one who feels this way and so, if they could be acted upon, my ideas still wouldn't be as, it's best to have things make sense to the overall majority as opposed to just one guy. All in all, I still enjoyed implementing these, especially the first one in this section so I'm happy.



```
SELECT c.customer_id, c.first_name, c.last_name
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL;
```

*Figure 5.2: Image of SQL code to check which customers have no orders.*



```
SELECT a.author_id, a.first_name, a.last_name
FROM Authors a
LEFT JOIN
Books b ON a.author_id = b.author_id
WHERE b.book_id IS NULL;
```

*Figure 5.3: Image of SQL code to check which authors have published no books.*

## Conclusion

So, what has been learned because of this project? Well, many things but the first is that I really don't like trying to find small mistakes! I already knew that to some degree as I, before I entered the IT Programming program, had quite a bit of prior experience in Java which can have some of the same issues. But, either due to their differences or due to my experience, I find myself making less mistakes in that language. So, it has been quite some time since I've felt such frustration. But, with such frustration and the drive to not give up, comes with great satisfaction once all is said and done. Of course, this is not a situation in which most would give up considering the stakes and relative ease of things, but I think it a nice thing to point out that it's good not to give up.

I also had quite a bit of fun creating the tables because of MySQL. As mentioned earlier, I didn't use the GUI to create the tables, but it was quite satisfying hitting that refresh button and then seeing my newly created table appear in the side menu. The same thing goes for when I inserted the data and then ran that o so important command; SELECT * FROM Table. Just running it and then having proof I had done something, that's one of the greatest things about SQL and programming in general. I think it's one of the reasons I like the craft so much, there's truly nothing like it. At least, nothing that evokes the same feeling in me. No one can see into the future, but we can sure as heck try, I feel as though this project will be a great steppingstone for my future. I much prefer programming but it's possible that I will go into database design and that's also completely ignoring the fact that there are things to be learned that apply to more than just database management such as organizational skills and things of that nature. As obvious as it may be since this is a school project, it truly was a learning experience for me. And, as I've said before, there were parts of this, what's supposed to be work, that I did enjoy. If I went into database management, I don't think I would be truly unhappy.

# Bibliography

Bhatta, R. (2011). *SQL Online Compiler*. Retrieved from Programiz: https://www.programiz.com/sql/online-compiler/

Hamedani, M. (2019, March 19). *Youtube, MySQL Tutorial for Beginners [Full Course]*. Retrieved from Youtube: https://www.youtube.com/watch?v=7S_tz1z_5bA&ab_channel=ProgrammingwithMosh

Ho, D. (2003, November 24). *Notepad++ Home Page*. Retrieved from Notepad++: https://notepad-plus-plus.org/

MySQL AB. (1995, May 23). *MySQL Home Page*. Retrieved from MySQL: https://www.mysql.com/

Refsnes Data AS. (1998). *MySQL SUBSTRING_INDEX() Function*. Retrieved from w3schools: https://www.w3schools.com/sql/func_mysql_substring_index.asp