MySql Tutorial

(createing foreign keys without using the data modeling tool)

Contents

1. Overview
2. Connect to Your Database
3. Check Your Customer and Product Table Design
4. Create an Associative Table (purchase) With Foreign Keys
5. Add Data to your Associative Table (Purchase)
6. Troubleshooting (Foreign Keys)
7. Reviewing Single Table Select Statement
8. Select Data from all Three Tables (SQL JOIN)
9. Summary
1. **Overview**

In a previous tutorial, you created and populated data into two tables (customer and product) in a MySQL database schema using MySQLWorkbench. In this tutorial, you will reverse engineer a data model from that schema.

“Reverse engineering” (generally) means creating a plan from an existing implementation, whereas “forward engineering” (more common) means making a plan first and then implementing from that plan. When you “reverse engineer” a database, it means you are creating a data model (picture of the database tables and their relationships) from an existing database.

Then, from within the data modeling tool, you will add a third table (purchase) that implements a many-to-many relationship (between customer and product). You will then synchronize the data model to your database schema, effectively adding the new table into the database schema, along with foreign key constraints.

You will enter some test data into this new table (purchase) and notice how the Database Mgt System enforces primary key and foreign key constraints. Then, you will create a SQL select statement that joins data from all three of these tables.

Hopefully, you will gain an appreciation for the concept of relational database design where no redundant data is stored. Since data is only stored in one place, web applications do not have to work hard to “update all copies”. Wherever users want to see data in multiple places, your web application only needs to execute a sql join to show data that is useful and descriptive – redundant data is shown, but not stored, in the database.

2. **Connect to Your Database**

Using MySQLWorkbench, connect to the database schema that has been created for you on cis-linux2. The name of your database schema will be in this format: SP16_2308_tua12345. Instructions for how to connect to your database (and select your database as the default database) are in a separate document.
3. **Check Your Customer and Product Table Design**

Check the design of your Customer and Product table. Recall that you do this by right clicking the name of your table (in the tree view in the left “object browser” pane) and selecting “alter table”. If you do not have 3-5 records in each table, enter some test data now.

![Customer table design](image)

**Product table design**

**Review the Abbreviations for the checkboxes:**

- **PK:** Primary Key (unique identifier of each record within the table)
- **NN:** Not Null (not allowed to be null, means it is a required field for the user to enter)
- **UQ:** values in this field must be unique within the table.
- **AI:** Auto-increment. The database management system supplies the next available number (automatically) – the user does not have to enter a value into this field.

Also, make sure each of your tables has the “InnoDB” Engine specified since we need this to support the foreign keys that we are about to add.

![SQL file showing InnoDB Engine](image)
4. Create an Associative Table (purchase) With Foreign Keys

The table that will implement a many-to-many relationship between customer and product is the purchase table. Each purchase record will indicate which customer did the purchasing and what product was purchased (along with other information like how many items were purchased and when).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>PK</th>
<th>NN</th>
<th>UQ</th>
<th>B</th>
<th>UN</th>
<th>ZF</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>purchase_seqno</td>
<td>INT(11)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purchase_qty</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purchase_date</td>
<td>DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>customer_id</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>product_id</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To have the DataBase Management System (MySQL) manage the integrity of the database, we will make the last two fields into Foreign Keys. Once we do this, the DBMS will not allow any invalid values for customer_id nor for product_id (in table purchase). To turn the customer_id into a foreign key, type the SQL into a Query window:

```sql
alter table purchase add foreign key (customer_id) references customer(customer_id);
```

Then alter the purchase table and notice that the customer_id icon changed to pink (to indicate that it is a foreign key field). Click on the “Foreign Keys” tab of the table design UI and see more information about the foreign key constraint that you just created:

![Foreign Keys](image)

*Figure out the SQL you need to run to make purchase.product_id also be a foreign key (to product.product_id) and then run that.*
5. Add Data to your Associative Table (Purchase)

Now that you have successfully placed Foreign Key constraints to the associative table (purchase table) in your database, the database management system (MySql) will not allow anyone to enter a record into that associative table unless it’s Foreign Keys point to valid (existing) records in the other tables. In the example above, this means no one can enter a purchase record unless the purchase.customer_id has a value that matches some customer_id in the customer table (and purchase.product_id matches some product_id in the product table).

Check to see what Primary Key (PK) values you have now in customer.customer_id and in product.product_id. These are the only values you are allowed to enter into the Foreign Key (FK) fields customer.customer_id and product.customer_id (in your associative table, purchase). Using MySqlWorkBench, try to enter a row into your associative table.

- First enter a record that has two valid FK values (a valid customer_id and a valid product_id), click the green arrow to apply the changes. See if it lets you insert the record (it should).
- NOTE: you must enter dates in this format: YYYY-MM-DD
- Next, try to add a record with a bad value for customer ID. Try to apply the changes and it should give you an error message, specifying a Foreign Key (FK) error that mentions your customer table.
- Next, try to add a record with a bad value for product id. When you try to apply the changes, it should give you an error message, specifying a Foreign Key (FK) error that mentions your product table.

Add 8-10 records into your associative table (purchase). To help you better understand what the associative table does, make sure to add more than one associative record for some customer and make sure to add more than one associative record for some product.

FYI: for some data models, it is desirable to allow certain Foreign Key (FK) fields to hold a null value. For example, a null value in student.major_id would represent a student who has not declared a major yet. Once you start allowing null values for foreign keys, you have to learn about “outer join” select statements. With an inner join, you can select all students along with their major – including those students who have null for major_id. If you do not use an “outer join”, your select statement will only show the students who have declared a major. This tutorial does not go into optional foreign key fields and it does not cover “outer join”, but it is an important topic and one that you should learn before going out to interview for a job.
6. **Troubleshooting (Foreign Keys)**

If you need to delete a Foreign Key constraint, you can

- right click on your purchase table and select "alter table" then click on the "Foreign Keys" tab
- select one or both Foreign Keys, then right click and select “delete”

Then, you can retry the exercise.

![Diagram of Foreign Keys options](image-url)
7. Reviewing Single Table Select Statement

Recall that if you want to show all of the columns and all the rows of a table (where order does not matter), you can simply run this query (type the SQL command into the query window at the top and then click the lightning/execute icon):

```sql
SELECT * FROM table_name;
```

![SQL select statement example](image)

If you want to show only certain columns and selected rows and specify a desired order, you can create a SQL select statement like this. The WHERE clause restricts which rows are shown.

```sql
SELECT last_name, first_name, city, state
FROM customer
WHERE state='pa'
ORDER BY last_name, first_name;
```

![SQL select statement example](image)

8. Select Data from all Three Tables (SQL JOIN)

The real power of a relational database is the ability to join data from related tables. Recall that two tables are “related” to each other if one has a foreign key that “points to” the primary keys in the other table. Your web application users will want to see more than just product IDs and customer IDs when they look at the purchase data. They will want to see the description and price of the product, and the name and address of the customer who purchased it. This is accomplished using a **SQL join**.
If the database has this data in it:

![Image of database tables]

Then to join the data from these 3 tables, you could use the following SQL select statement:

```sql
SELECT * FROM customer, purchase, product
WHERE customer.customer_id = purchase.customer_id
AND product.product_id = purchase.product_id;
```

And you would get this result set (there are additional columns to the right that didn’t fit in the screen capture).

![Screenshot of result set]

The above result set shows each purchase record along with the product that was purchased as well as the customer who bought the product. Note that wherever a column name is ambiguous (such as customer_id which exists in both the customer table as well as the purchase table), you must prefix the column name with the table name in your SQL statement.

To save on typing, you can assign table aliases as shown. For example, customer was given the alias C so that “C” can be used instead of the longer name “customer”.

Sally Kyvernitis, Temple University, Page 8
SELECT * FROM customer AS C, purchase AS PU, product AS PR
WHERE C.customer_id = PU.customer_id AND PU.product_id = PR.product_id;

To understand a little better what’s going on with the join, consider a statement like “select * from customer, purchase” (where you are selecting data from two tables but did not filter out unwanted rows using the WHERE clause). You get what is called the “cross product” which is basically useless “white noise”. It shows every row in the first (customer) table along-side every row in the second (purchase) table. So, this result set is functionally useless since it shows every purchase next to every customer who may or may not have done the purchasing. Since there were 4 purchase records and 3 customers, it shows 12 rows (3*4). The first row in purchase with each of 4 customers, the second row in purchase with the same 4 customers, and so on.

SELECT * FROM customer, purchase;
In order to make the data useful, you must restrict the select statement to ONLY display see those joined records where the purchase.customer_id matches up with a customer.customer_id (as was done initially).

Here is an example of a useful SQL SELECT statement that joins data from all 3 of your tables. This is the kind of select statement that might supply data for a page of your web application. Note the calculated columns (which are also aliased) and the ORDER BY clause which is always important when displaying information to users.

```sql
SELECT last_name, first_name, purchase_seqno, purchase_date, PUR.product_id, product_desc, retail_price, purchase_qty*retail_price as extended_price
FROM customer AS C, purchase AS PUR, product AS PR
WHERE C.customer_id = PUR.customer_id AND PUR.product_id = PR.product_id
ORDER BY last_name, first_name, purchase_date;
```

9. Summary

In part 1 (the previous) tutorial, you used MySqlWorkbench (GUI front end) to connect to MySql (DBMS - DataBase Management System) and create two (unrelated) database tables. You added data into these tables and executed single table SQL SELECT statements.

In part 2 (this tutorial), you used MySqlWorkbench to accomplish the following tasks.

- Created a data model by reverse engineering from your database. The model contained the tables that were already in the database (customer and product).
- Added an associative table (purchase) that implemented a many-to-many relationship between two other tables (customer and product). This was done by creating a one-to-many relationship of product to purchase (one product can have many purchases) and a one-to-many relationship of customer to purchase (one customer can have many purchases). For each relationship that was added, a Foreign Key was created. The foreign keys were: purchase.customer_id and purchase.product_id.
- To understand the concept of a foreign key constraint, you tried to enter a record into your associative table (purchase) that had an invalid foreign key, e.g., a purchase record with an invalid customer_id, a purchase record with an invalid_product_id. You found that the DBMS rejected a purchase record if there was no matching customer_id in the customer table and/or if there was no matching product_id in the product table.
- Created a SQL select statements that joined data from all 3 related tables in your database, essentially showing the associative table (purchase) records along with descriptive attributes from its related tables (“customer” and “product”).

Hopefully, these two tutorials demonstrated the power of a relational database design where no redundant data is stored (and consequently having no data inconsistencies). Your web application will only have to update data in one place, not many places (which would be much more work and error prone). Wherever your app needs to show redundant data you will use a Sql SELECT statement to join data from multiple related tables.