# Databases Lab 4 (MySQL)

We are going to look at creating tables and populating them with data.

The scenario is a basic order processing system.

## Scenario

A company requires an order processing system. Customers place Orders which can be for one or more Stock Items.

Customers have ids, names, and email addresses. For a Stock Item, an id, description, quantity on hand and unit price is known.

An Order has an id and an order date and identifies the customer id and a set of stock items together with the quantity being ordered.

## Exercise

Ensure you understand the application of **Entity Relational Modelling** to this scenario as described below:

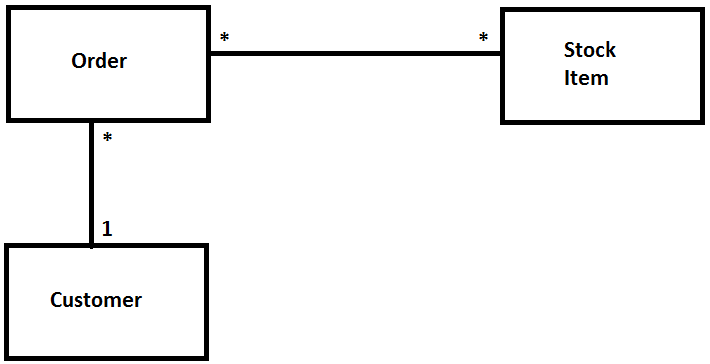
1. Revisit the scenario description and identify entities and attributes through considering the nouns:

A company requires an order processing system. **Customers** place **Orders** which can be for one or more **Stock Items**.

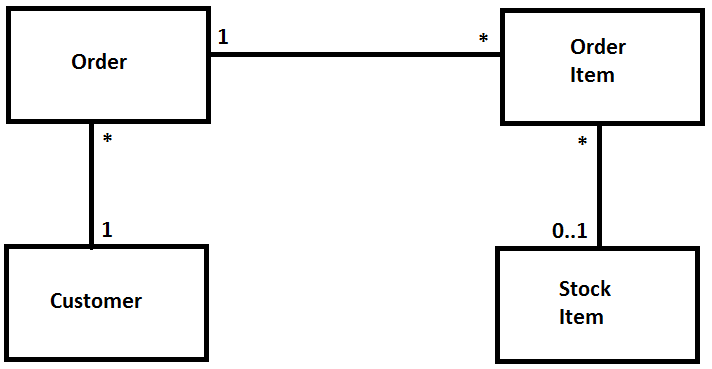
Customers have *ids*, *names*, and *email addresses*. For a Stock Item, an *id*, *description*, *quantity on hand* and *unit price* are known.

An Order has an *id* and an *order date* and identifies the *customer id* and a set of stock item *ids* together with the *quantity* being ordered.

1. This gives us an initial data model like this:



1. Recall that the Relational Model dislikes many-to-many relationships (M:N) and that we resolve this by introducing a link entity: in this case, **Order Item** where an **Order** has many **Order Items** and a **Stock Item** can appear in many **Order Items**. This gives a Data Model like this:



Note the 0..1 indicator at the **Stock Item** end as a **Stock Item** may not yet have been ordered.

1. Next we can write down the relations and identify primary and foreign keys:

Customer(CustomerId, Name, Email)

Order(OrderId**,** OrderDate, CustomerId)

OrderItem(**OrderId**, **StockId**, Quantity)

StockItem(StockId, Description, QuantityOnHand, UnitPrice)

Recall that a Primary Key attribute (set of attributes) uniquely identifies an entity instance e.g. a **Customer**, an **Order**, an **Order Item**; while a Foreign Key attribute (set of attributes) establishes a relationship with another relation e.g. *CustomerId* in **Order**; this links an order instance with a customer instance.

## Exercise

Ensure you understand the application of **Normalisation** to this scenario as described below:

1. If we examine the scenario, we can establish a single relation which incorporates each of the pieces of information the system requires to persist:

Order(OrderId, OrderDate, CustomerId, Name, Email,

(StockId, Description, QuantityOnHand, UnitPrice, Quantity)

)

We have identified *OrderId* as the Primary Key of this single relation. Note within each **Order** we have a repeating group identifying the **Order Items** attached to a specific **Order**.

1. First Normal Form (1NF) removes repeating groups and forms new relations. Let’s remove the **Order Item** repeating group:

Order(OrderId, OrderDate, CustomerId, Name, Email)

OrderItem(OrderId, StockId, Description, QuantityOnHand, UnitPrice, Quantity)

We have created a new **OrderItem** relation with *OrderId* and *StockId* as the compound Primary Key.

1. Second normal form (2NF) examines partial key dependencies and the **Order** relation is already in 2NF as it is has a single key attribute.

Let’s consider the **OrderItem** relation and note that the *description*, *quantityonhand* and *unitprice* are dependent only on the *StockId* partial key attribute as they will be the same for that **Stock Item** each time it’s ordered (we are avoiding the complication of the unit price of a stock item changing). We can then remove these attributes to a new relation – **Stock Item**.

Order(OrderId, OrderDate, CustomerId, Name, Email)

OrderItem(OrderId, StockId, Quantity)

StockItem(StockId, Description, QuantityOnHand, UnitPrice)

1. Third normal form (3NF) concerns non-key dependencies; examining each non-key attribute in turn we see that *Name* and *Email* are functionally dependent on the non-key attribute *CustomerId* as *CustomerId* maps to single values of *Name* & *Email*. We extract these attributes to a new relation. This gives us a final set of relations:

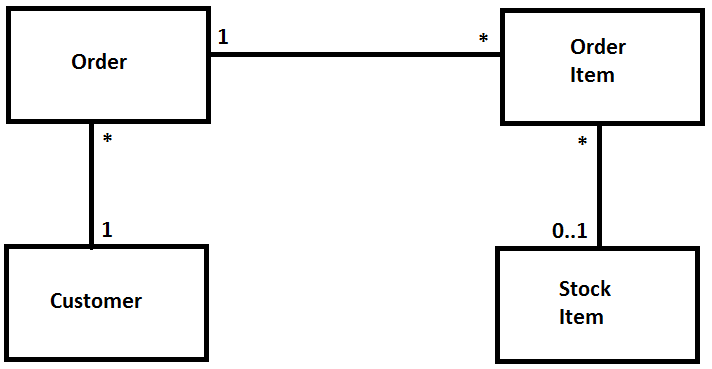
Customer(CustomerId, Name, Email)

Order(OrderId**,** OrderDate, CustomerId)

OrderItem(**OrderId**, **StockId**, Quantity)

StockItem(StockId, Description, QuantityOnHand, UnitPrice)

1. **Normalisation** has produced a similar result to the **Entity Relationship Modelling** approach:



## Exercise

We can now look at writing SQL statements to create the various tables.

### Customer

**CREATE TABLE CUSTOMERS**

**( CUSTOMER\_ID INT NOT NULL,**

**NAME VARCHAR(50) NOT NULL,**

**EMAIL VARCHAR(50),**

**PRIMARY KEY (CUSTOMER\_ID)**

**);**

Note this identifies a Primary Key constraint and Not Null constraints for those columns where a value must be supplied at the time of inserting a new customer row into the **Customers** table – we are allowing *Email* to be supplied at a later date.

### Stock Item

**CREATE TABLE STOCKITEMS**

**( STOCK\_ID int NOT NULL,**

**DESCRIPTION VARCHAR(50) NOT NULL,**

**QUANTITY\_ON\_HAND INT(4) NOT NULL,**

**UNIT\_PRICE DECIMAL(5, 2) NOT NULL,**

**PRIMARY KEY (STOCK\_ID)**

**);**

Again, we have a primary key – *StockId*, and, not NULL constraints.

### Order

**CREATE TABLE ORDERS**

**( ORDER\_ID INT NOT NULL,**

**CUSTOMER\_ID INT NOT NULL,**

**ORDER\_DATE DATE NOT NULL,**

**PRIMARY KEY (ORDER\_ID),**

**CONSTRAINT ORDERS\_FK**

**FOREIGN KEY (CUSTOMER\_ID) REFERENCES**

**CUSTOMERS(CUSTOMER\_ID)**

**);**

Note the Foreign Key constraint to create the relationship between **Order** and **Customer** i.e. a one-to-many relation where a Customer row can match to 0 or many Order rows.

### OrderItem

**CREATE TABLE ORDERITEMS**

**( ORDER\_ID INT NOT NULL,**

**STOCK\_ID INT NOT NULL,**

**QUANTITY INT(4) NOT NULL,**

**PRIMARY KEY (ORDER\_ID, STOCK\_ID),**

**CONSTRAINT ORDERITEMS\_ORDER\_FK**

**FOREIGN KEY (ORDER\_ID) REFERENCES ORDERS(ORDER\_ID),**

**CONSTRAINT ORDERITEMS\_STOCKITEM\_FK**

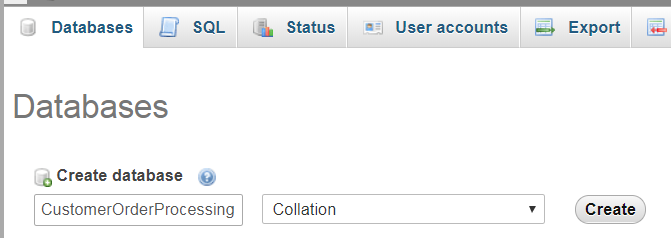
**FOREIGN KEY (STOCK\_ID) REFERENCES STOCKITEMS(STOCK\_ID)**

**);**

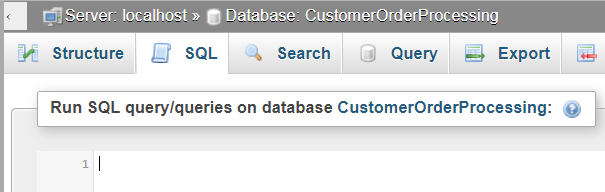
Note the foreign key constraints to provide the link table between **Order** and **Stock Item.**

## Exercise

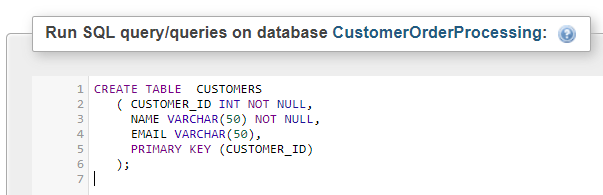
Launch **phpMyAdmin** as before, navigate to the **Databases** tab and create a new database with the name *CustomerOrderProcessing* as below:



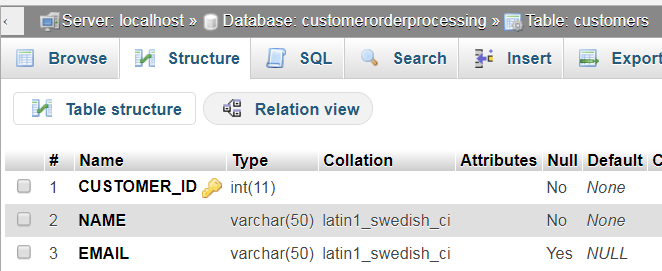
1. Select the **SQL** tab:



1. Enter and run the **Create Table** statement for the **Customers** Table:



Navigate to the Customers table **Structure** tab to check the table definition:



1. Enter and run the **Create Table** statements for each of the other tables. Note you will have to use the pattern **StockItems**, **Orders**, **OrderItems** to ensure the foreign key constraints can be created properly.

## Exercise

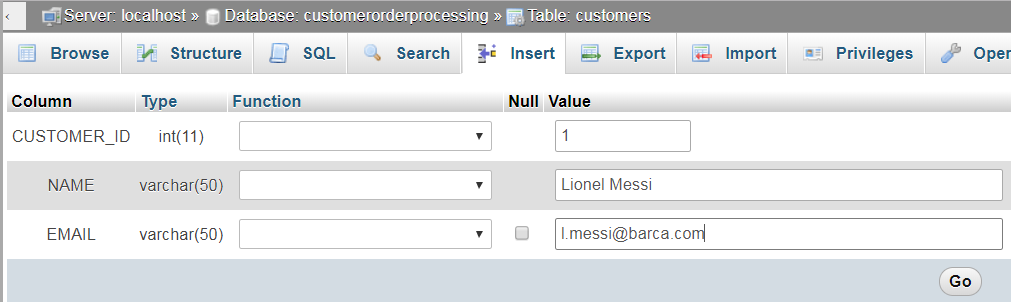
Now we are going to populate the tables with sample data. **phpMyAdmin** provides facilities for inserting new rows.

1. Navigate to the **Customers** table, click on the **Insert** tab to be given a form to input new **Customer** details. Enter the data below:

**CustomerId: 1**

**Name: Lionel Messi**

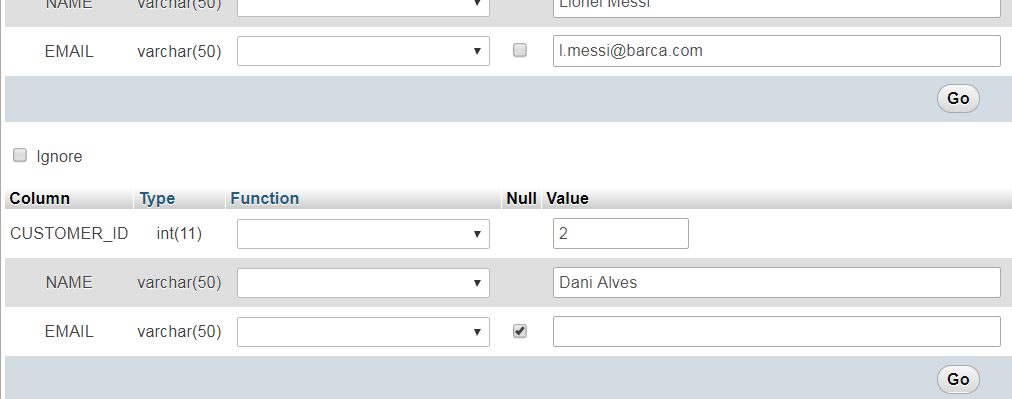
**Email: l.messi@barca.com**



Repeat with the following details in the second form:

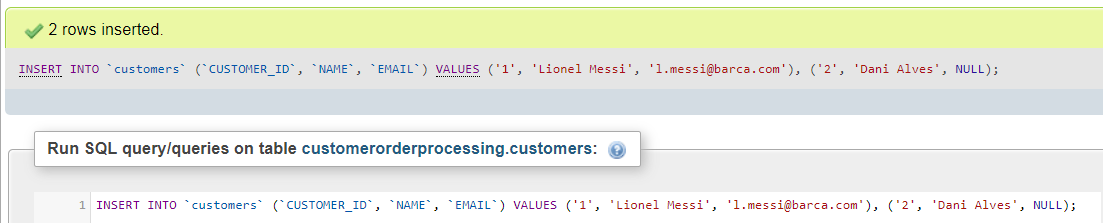
**CustomerId: 2**

**Name: Dani Alves**



Click the **Go** button on the second form to insert both rows.

Note the SQL statement which has been executed:



1. Use the facilities to enter the data from the following data set:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Order Id** | **Order Date** | **Customer Id** | **Name** | **Email** | **Stock Id** | **Description** | **Quantity On Hand** | **Unit Price** | **Quantity** |
| 1 | 2016/02/15 | 1 | Lionel Messi | l.messi@barca.com | 1 | Bluetooth Cap - Winter Beanie Hat with Bluetooth Stereo Headphones, Microphone, Hands Free System and Rechargeable battery | 25 | 10.00 | 1 |
| 1 | 2016/02/15 | 1 | Lionel Messi | l.messi@barca.com | 2 | TP-LINK TL-SG108 8-Port Metal Gigabit Ethernet Switch | 52 | 13.99 | 1 |
| 2 | 2016/02/15 | 2 | Dani Alves |  | 1 | Bluetooth Cap - Winter Beanie Hat with Bluetooth Stereo Headphones, Microphone, Hands Free System and Rechargeable battery | 25 | 10.00 | 2 |
| 3 | 2016/02/16 | 1 | Lionel Messi | l.messi@barca.com | 3 | BT Dual-Band Wi-Fi Extender (Booster) 600 | 10 | 27.99 | 2 |
| 4 | 2016/02/16 | 3 | Andres Iniesta | andyiniesta@hotmail.com | 4 | Asus GL552JX 15.6-Inch Laptop Notebook | 5 | 799.95 | 1 |

## Exercise

Write SQL statements for the following, checking the results using **SELECT** statements or via the **Browse tab**:

1. An INSERT statement to add a new customer

Name: Sergi Busquets

1. INSERT statements to enter new Stock Items as below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Stock Id** | **Description** | **Quantity On Hand** | **Unit Price** |
| 5 | VicTsing New USB 7 Buttons Wireless Professional Game Gaming Optical Mouse | 22 | 7.99 |
| 6 | 8" Inch Windows 10 Tablet PC | 25 | 109.99 |
| 7 | Lenovo Ideacentre Stick 300 Ultra Small PC | 12 | 99.99 |
| 8 | Logitech ConferenceCam Connect 1920 x 1080 Video Conferencing Camera | 87 | 259.99 |

1. An **INSERT** statement to add a new order for *Sergi Busquets* on *17th February 2016* for the *Gaming Mouse* and the *Conferencing Camera*.
2. An **UPDATE** statement to update a customer’s email address

Email: sergi.busquets@barca.com

For customer *Sergi Busquets*.

1. A **DELETE** statement to delete the order item of the *Conferencing Camera* for the new *Sergi* *Busquet’s* order.
2. A **SELECT** statement to list order details with items cost for a specific order.