Introduction to SQL with MySQL and Workbench

## Day 01

The world is made up of nouns. Nouns like bird, employee, account, firm, vehicle, student. These nouns can be described using adjectives such as name, weight, height, and colour. All nouns give rise to adjectives. In the database world, these nouns are referred to as entities, and their adjectives are their attributes (properties). These attributes then give rise to values. We capture all this information in records. Entities sometimes relate to other entities. For example a Student relates to a Course which relates to a Classroom. A student takes a course in a classroom. It is our jobs as database people, to make sure this capture of a *domain* is accurate and does not create problems. Everything must run smoothly. Therefore we have to organize the capture and dissemination of data. We use tools like databases, tables and records to accomplish all of this.

**Content for Day01**

**Part 1 – Environment Setup**

**Part 2 – Working with Tables**

**Part 3 – SQL Based Files**

**Part 4 – Editing Records**

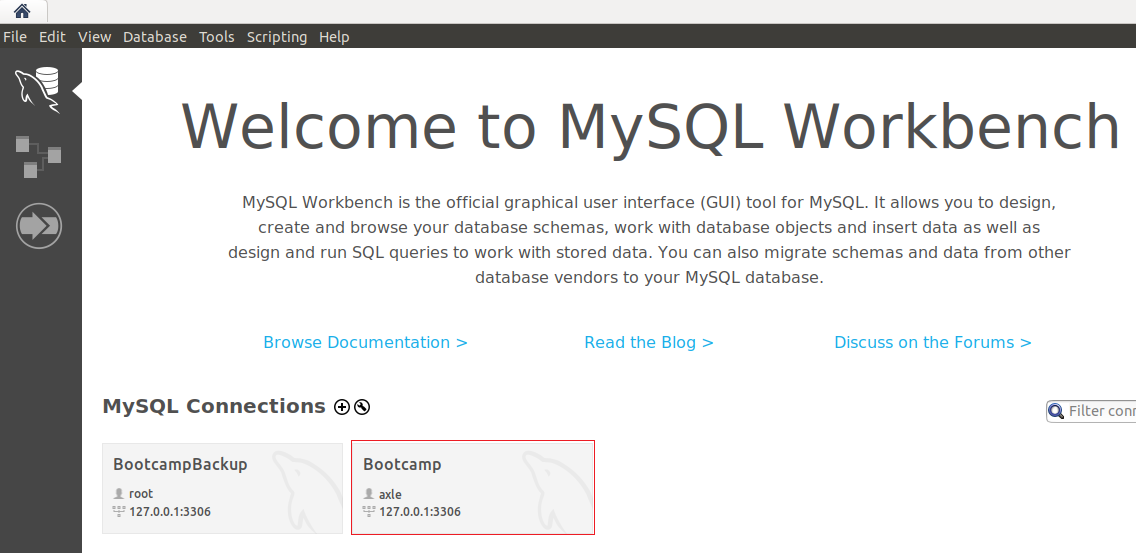
**Part 5 – Exploring SQL**

**Part 6 – Introduction to Constraints**

**Part 7 – Database & Table Creation**

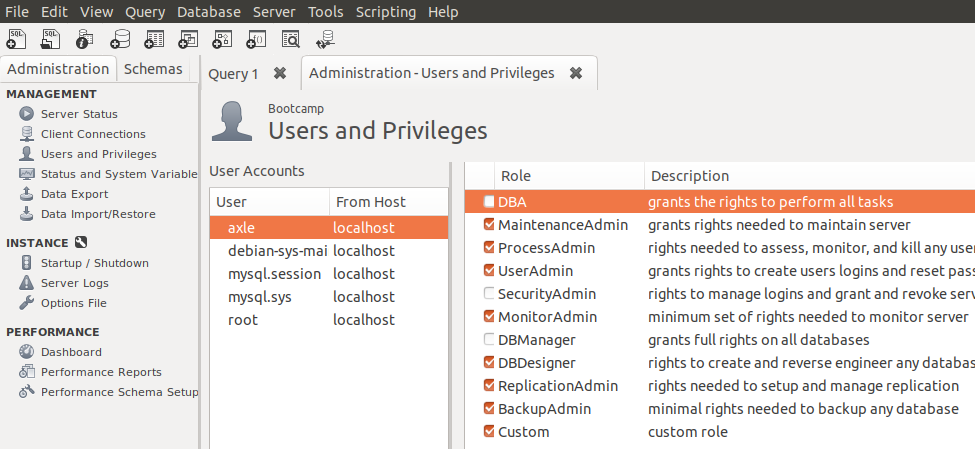
# Part 1 – Environment Setup

This Bootcamp assumes that you have the MySQL Server software installed. You should also have MySQL Workbench configured with **you** as the root user or a user with admin privileges. You should also have a connection to the database already established. In my case I am using an Ubuntu Linux Virtual Machine with the 14.14 version of MySQL and version 8 of MySQL Workbench.

1. In this first image, I have setup a Connection to the database server, from MySQL Workbench called simply *Bootcamp*. As you can see *axle* is the user. I also have a backup connection in case *axle* fails.

My suggestion is to create a different user, other than *root*, in order to be the “regular use” DBA. Refer to the Appendix A for details on how to create and promote a user.

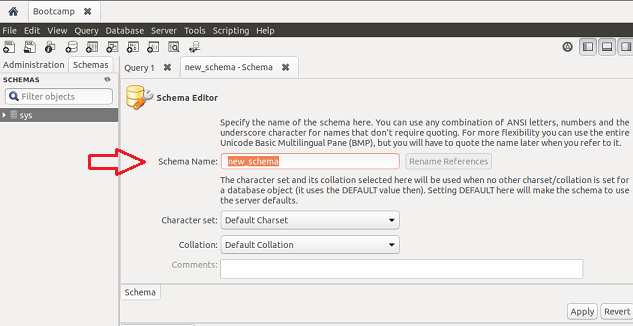
1. In this second image, you see the privileges of *axle*. I have set him to be DBA, so all administrative privileges:



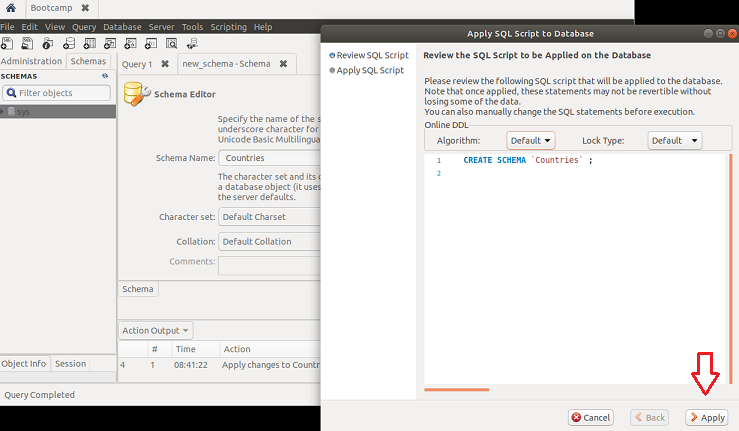
Ther should be two tabs on the left window of the UI, Administration and Schemas. This image shows the Administration tab, Schemas is for creating databases and working with tables. If you wanted to make *axle* a DBA, check Appendix A.

1. Graphical user interface, text, application

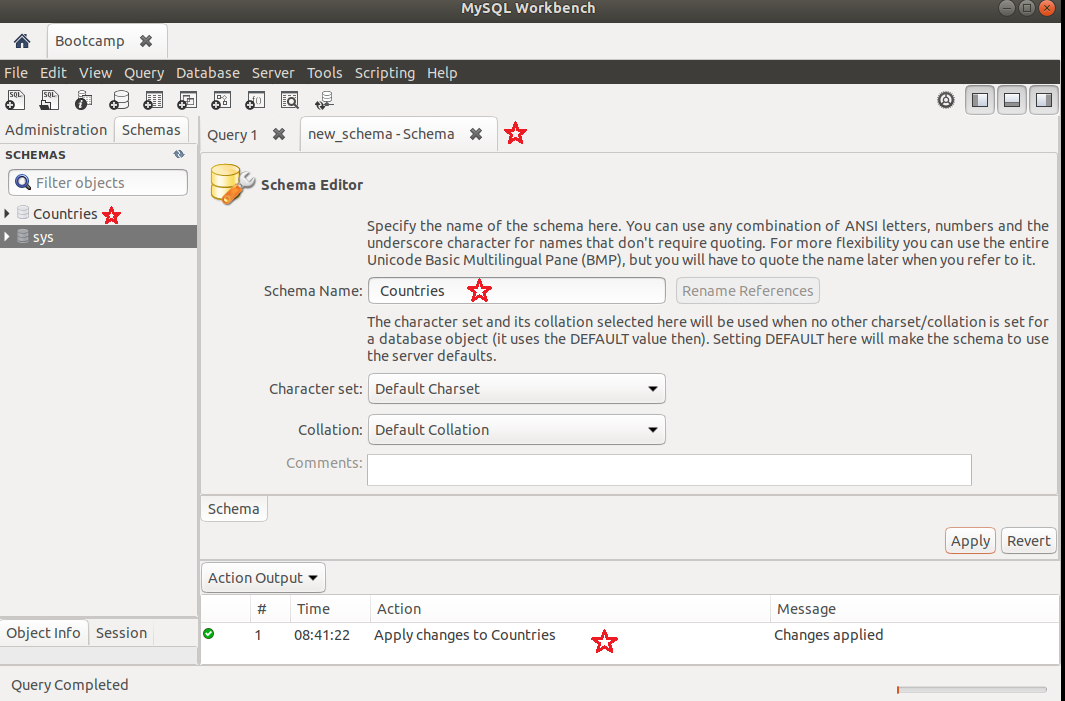
   Description automatically generatedYou will notice a Query 1 tab to the right of the Administration tab. This is where all the action is. It is this window(s) that we use to interface with the underlying MySQL database. Notice that there are several icons, just hover over each to see what they do. For example, in the image below, the mouse is hovered the icon.   
   It is ok if you do not understand what the words in the pop-up mean, meanings will come with practice.

1. If all goes well, we can now create a new database using the UI. This button  will create a database, just follow the prompts:  
   

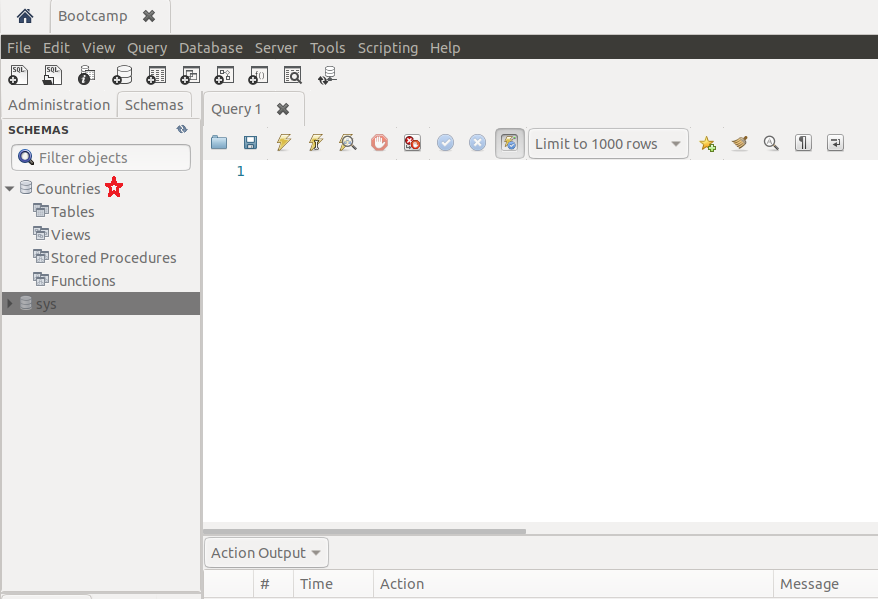
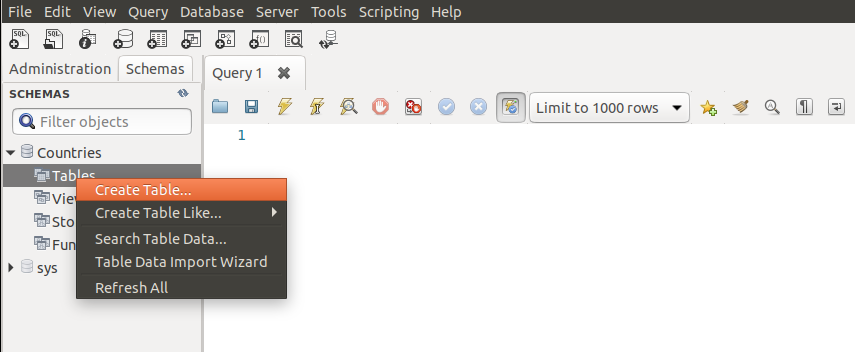
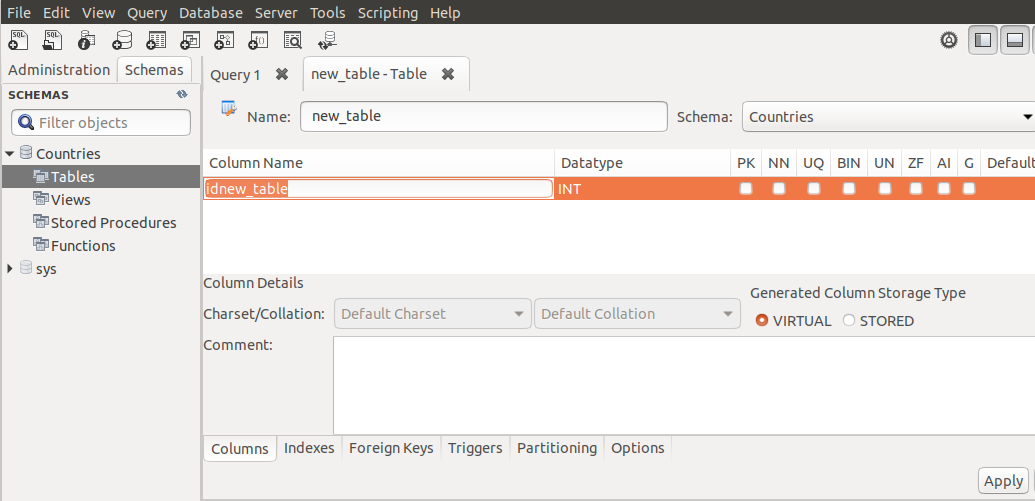
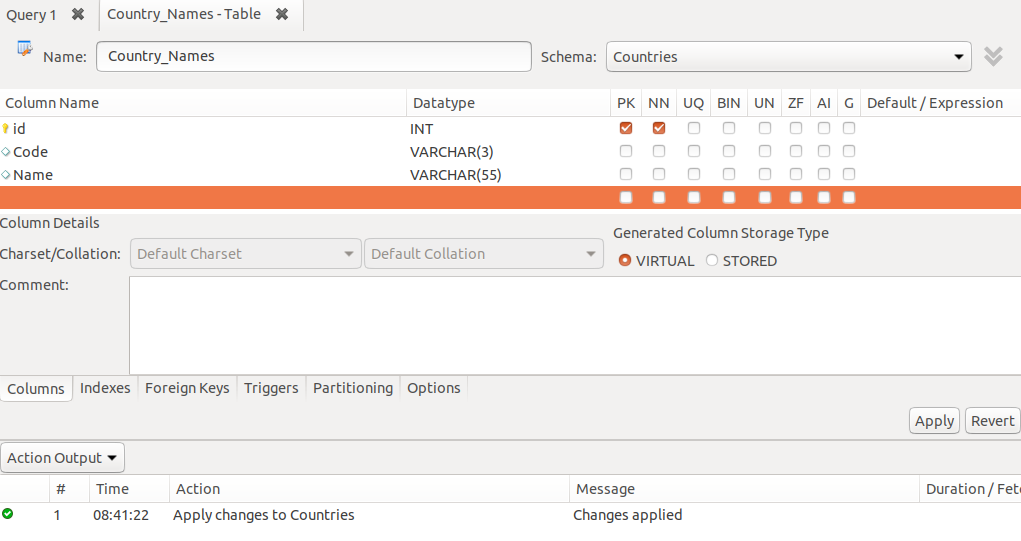
Once you click the button , a new dialog box opens up and asks for the name of the database (Schema). Lets call this one *Countries*. After you replace the orange text with the word ‘Countries’, click the Apply button down at the bottom right.

1. A second dialog box will open and ask you to confirm your decision. It even shows you the SQL statement that is being used to create the database for you. All you have to do is confirm by clicking on the Apply button:

The UI will respond with a confirmation page of its own, just clik on the >Close button.

1. So, back on the main UI, we are greeted with several important bits of feedback and confirmation of our actions. I have put red stars on the most important places. On the left we now have a new database, *Countries*. Up top the new\_schema\_Schema dialog tab remains and the Schema Name is showing ‘Countries’ as well. This tab can now be closed, but before you close it, notice that at the very bottom of this dialog page there is the Action Output tab. This tab is very important as it confirms almost all the actions you take using the UI.

Click the  to close the new\_schema\_Schema tab.

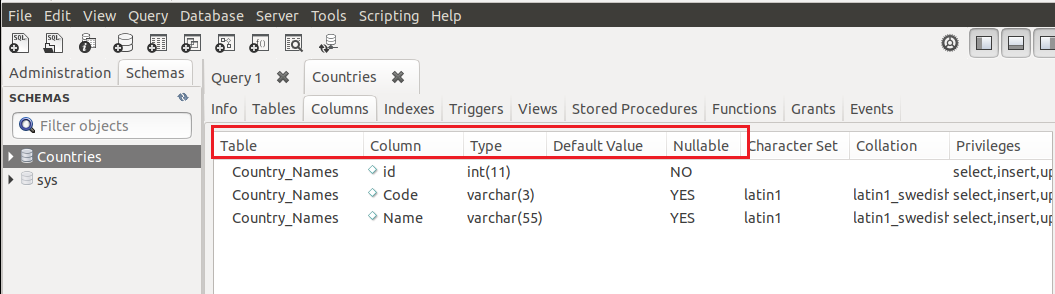
1. If you have the Schemas tab selected (not Administration), on the left pane of the UI, you should see two items, the *Countries* and Sys databases. Click on the drop down arrow to the left of the word ‘Countries’ and a new list of four sub-categories will expand out. We will use the Tables option to create a new table called *Country\_Names*:  
   
2. If you right click on Tables, you will get options related to Tables, just choose Create Table:
3. You will then see a dialog box that lets you describe your tables. Your table will be given a default name, *new\_table*, just chang it to *Country\_Names* and move to changing the column names also:  
   
4. Each table will have one or more columns. Each column will need a name and the type of data it will store, so *words*, *numbers*, *dates* etc. We will create two columns but the first column is special. The UI will insist that you create a numeric field and most times we can just name this column *id*. If you do not see the orange horizontal line, just double-click where you think it should be and it should return. Notice that the first column, *id*, has a few options. Tick PK and NN. We will return to what this means later, for now just continue to create two more columns one for the *country code* and one for the *name* of the country. In the Datatype column, you can click the mouse directly in that field to change the number of spaces that column uses, so 3 for *Code* and 55 for *Name*:  
     
   

Note: You will get an option to save your changes if you try to close this tab, or simply click the Apply button down the bottom right of the tab. Either way, you will get that confirmation dialog box, just click the Apply button on that box if you are comfortable with your changes.

|  |
| --- |
| End of section |

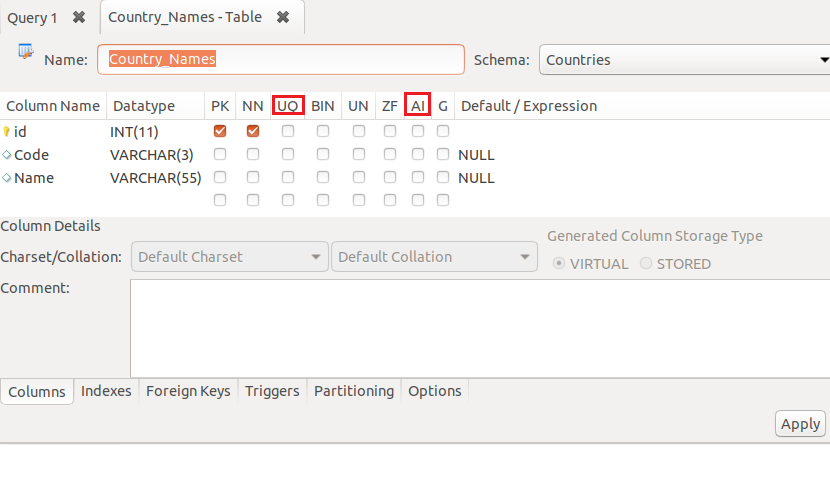
# Part 2 – Working with Tables

In this section we take a look at adding data to our tables. Along the way we discover more features of the UI

1. In this next demo, close all tabs in the main window except the Query 1 tab. Right-click on the *Countries* database on the left pane and choose the Schema Inspector option. You will now see several columns, they are all related to the database we created in Part 1. If you click on the Columns tab, you will see the three columns we created earlier:

For this boot camp we are only interested in the first 5 properties of each table. The other properties to the right are advanced features and for now we ignore them.

1. Close the Countries tab, click on the arrow to the left of the word Countries in the Schemas tab. Now the Tables option has an arrow next to it, click on that as well. Now right-click on Country\_Names and choose the Alter Table option:

  
As you can see the tab shows details about the table itself. In Part 1 we made the id column, the PK (Primary Key). It should also not be a null value (NN). It must exist. Now we will check the UQ (Unique) box and the AI (Auto Increment) box. Once you do this click the Apply button at the bottom right. Confirm your choices and close.

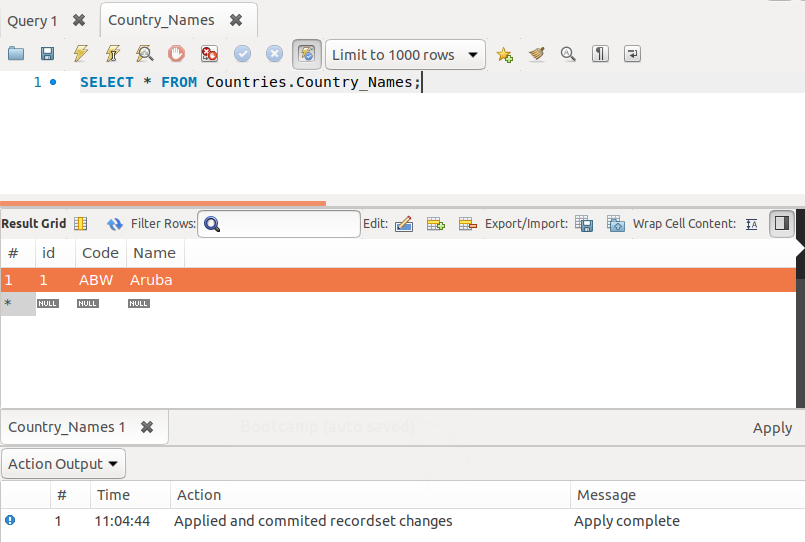
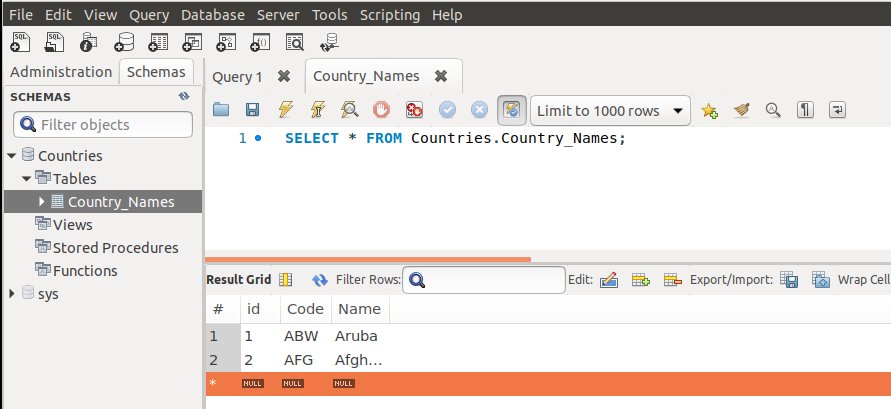
1. If your Tables -> Country\_Names is still available, right-click on Country\_Names and choose the Select Rows option:

Graphical user interface, text, application, email

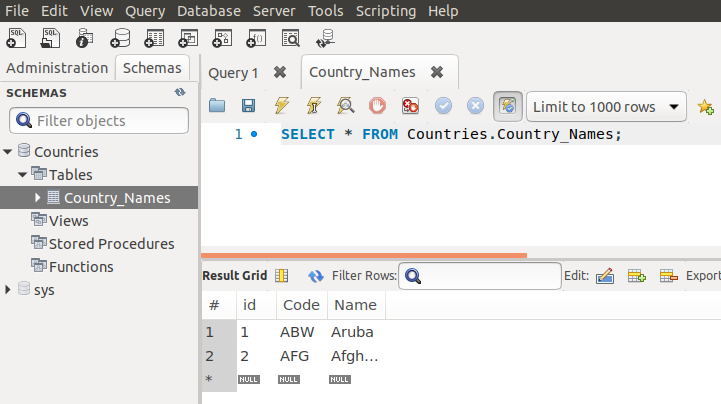
Description automatically generated  
As you can see the UI shows the SQL statement it ran just because of your actions. It also shows a Result Grid since you asked to see the first 1000 rows of data.

It just so happens that we do not have any data in the column fields as yet.

Finally at the bottom of that pane, there is the general feedback about your actions on the UI, were they successful or not.

1. The Result Grid is where we will start. Notice that there are three columns, the same ones we created in Part 1. Click on Code and type in the value ‘ABW’, then move to the Name column and enter ‘Aruba’. Once complete, just tab away:
2. Continue entering the next country which has a code of ‘AFG’ and the name of ‘Afghanistan’:

Remember to click the Apply button at the bottom right of that main window. At this point we have two records, each with unique IDs.

1. You can close the Country\_Names tab then Right-Click on Country\_Names located in the Schemas window and choose the Select Rows option again. This time we see the SQL statement the two rows of data we have:

|  |
| --- |
| You can now delete the table by right clicking on the table name in the Schemas window and choosing the Drop Table option. Truncate Table will simply remove all data from the table, it will not delete the table. You will be prompted of course, choose Drop Now option. Close the Country\_Names tab in the main window.  End of section |

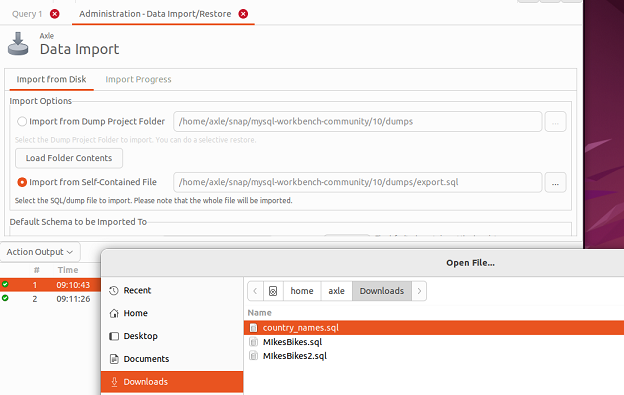
# Part 3 – Using SQL Files

In this section we take a look at using prepared SQL files. These files have everything we need, all we have to do is import the data. We can choose to build the table using the information in the file or just import the data. At the same time if we create tables from scratch, we can then export those tables for someone else to use. The file we will use here was converted to the UTF-8 character set.

1. You will have a file called country\_names.sql laying in a folder somewhere. First open that file in any text editor, its just text, and look at the SQL statements:

|  |
| --- |
| **DROP TABLE IF EXISTS `country\_names`;**  **/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;**  **/\*!40101 SET character\_set\_client = utf8 \*/;**  **CREATE TABLE `country\_names` (**  **`Code` CHAR(3) NOT NULL DEFAULT '',**  **`Name` CHAR(52) NOT NULL DEFAULT '',**  **`Continent` enum('Asia','Europe','North America','Africa','Oceania','Antarctica','South America') NOT NULL DEFAULT 'Asia',**  **`Region` CHAR(26) NOT NULL DEFAULT '',**  **`SurfaceArea` FLOAT(10,2) NOT NULL DEFAULT '0.00',**  **…**  **`Code2` CHAR(2) NOT NULL DEFAULT '',**  **PRIMARY KEY (`Code`)**  **) ENGINE=InnoDB DEFAULT CHARSET=latin1;**  **/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;**  **--**  **INSERT INTO `country\_names` VALUES ('ABW','Aruba','North America','Caribbean',193.00,NULL,103000,78.4,828.00,793.00,'Aruba','Nonmetropolitan Territory of The Netherlands','Beatrix',129,'AW');** |

This is an edited version of the file that you have. Notice that the file has comments and actual SQL code. Each statement will be executed in order, just like computer code. So first the table will be deleted if it is found, then a new table will be built and finally data will be added.

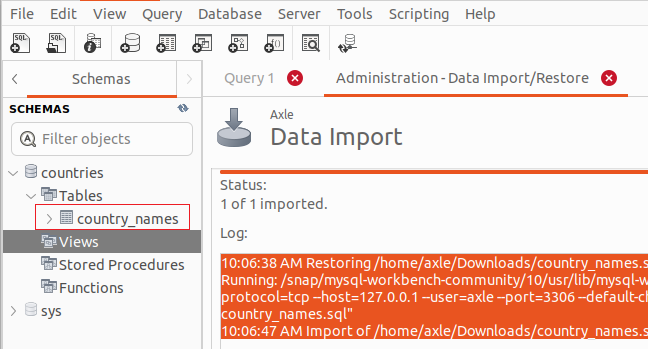
1. There are multiple ways to import data, for this part we will import via the menu system of Workbench. Go to Server, then option Data Import. We will import from a self-contained file. Use that dialog box to search and locate your country\_names.sql file. Once found, click on the green Open button on the top right corner of the dialog box:

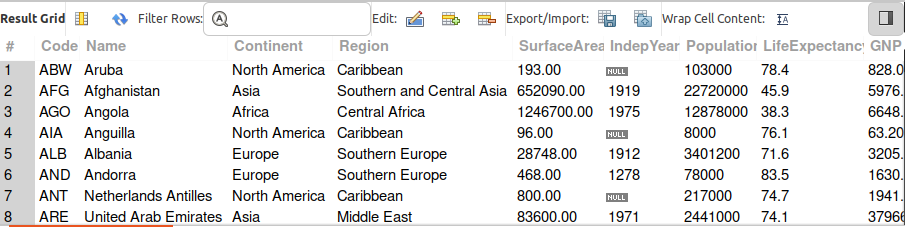
Under the Import Options, choose the second option Import from Self-Contained File. Once you click on the three elipses, a dialog box will open and you can search for your file.

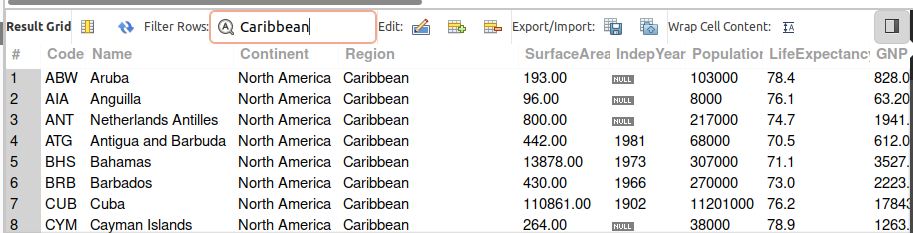
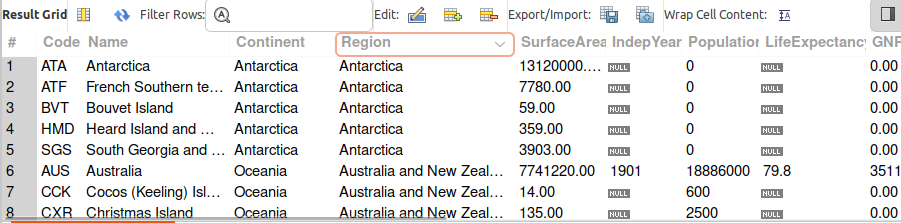
If you get and Unknown File Encoding dialog box, choose latin 1 for the Character Set Encoding Name and then click OK. Note also, I had to change the name of the database in the .sql file to *countries* instead of *Countries*.

1. If you got this far, remember to click on the Start Import button at the bottom right of that window. You may have to scroll a bit to get to it. If the import goes well, you should see a message like this one in the window:

|  |
| --- |
| **10:06:38 AM Restoring /home/axle/Downloads/country\_names.sql**  **Running: /snap/mysql-workbench-community/10/usr/lib/mysql-workbench/mysql --defaults-file="/tmp/tmpt6remfpf/extraparams.cnf" --protocol=tcp --host=127.0.0.1 --user=axle --port=3306 --default-character-set=utf8 --comments < "/home/axle/Downloads/country\_names.sql"**  **10:06:47 AM Import of /home/axle/Downloads/country\_names.sql has finished** |

1. If you see that message, click on the refresh button, it is a little icon  next to the word SCHEMAS in the left window pane. You should now see the country\_names table there:
2. Right-click on the country\_names table name and choose the option to Select Rows – Limit 1000. This will retrieve all the records. There are about 239 countries in the world so this is the number you should see in the **Action Output** window:



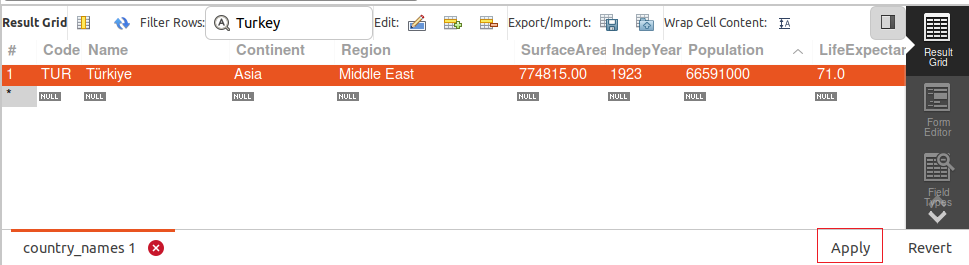
1. Notice the Filter Rows input box, enter Caribbean into the box and you will see that only countries in the Caribbean regeion will show up:  
   
2. Remove the word Caribbean from the search box and hit the enter key. You can also sort the various columns. For example if you click the Region header field, you will sort the rows by region:  
     
   

Also notice the faded arrow inside the Region header, that can be toggled to sort ascending or descending.

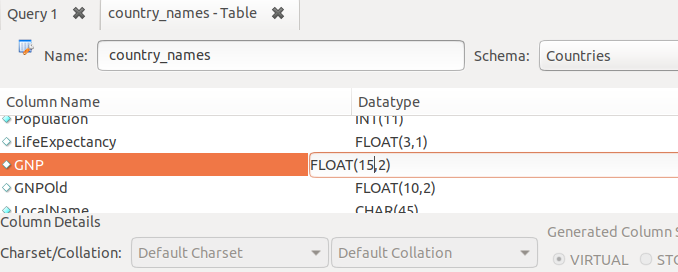
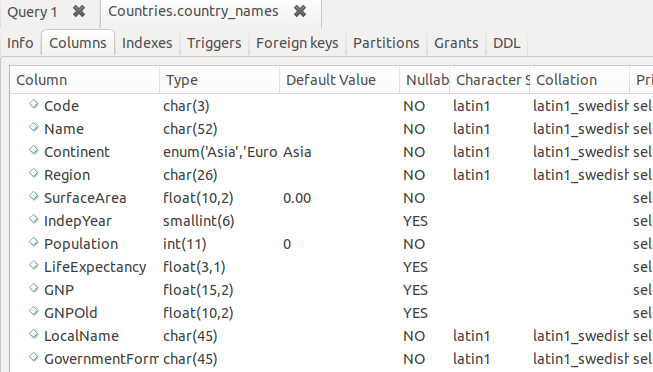
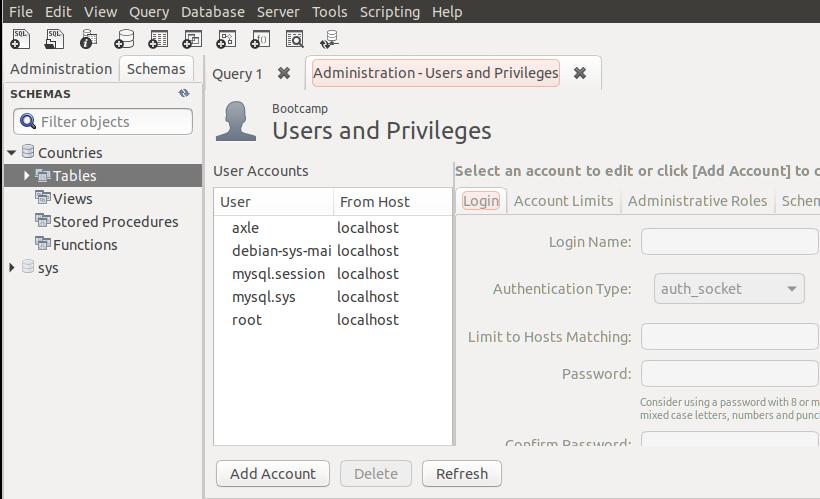
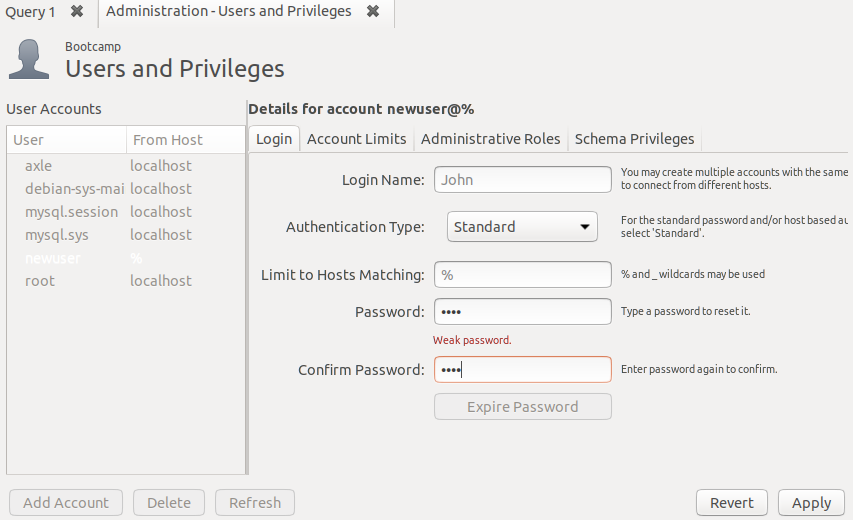
|  |
| --- |
| End of section |

# Part 4 – Editing Records via the UI

The country\_names table is very old and should not be used professionally. However for our purposes we can use it for learning opportunities. Note, in this part, we will be changing the name of the country of Türkiye formally known as Turkey. There are several ways to get the ü in the name Türkiye, but the easiest way is to just copy and paste from an existing letter. Another option is using the character map in your operating system.

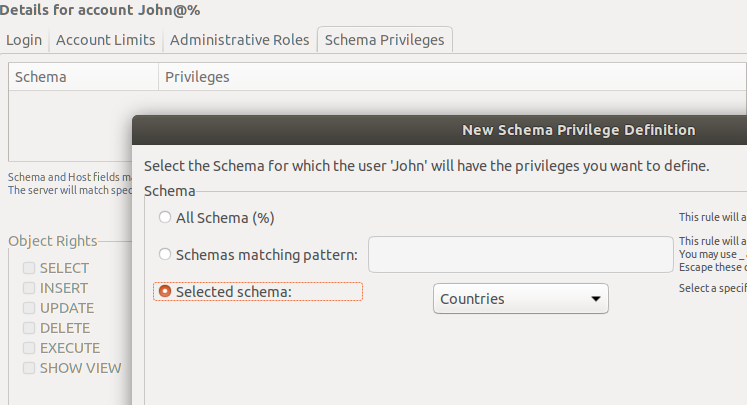
1. Actually in order to edit any of the fields, all we have to do is rest our mouse pointer on the field and click. Remember the search box called Filter Rows. We can use it to go to the country of *Turkey*. Change the name to *Türkiye* and remember to clik on the Apply button.

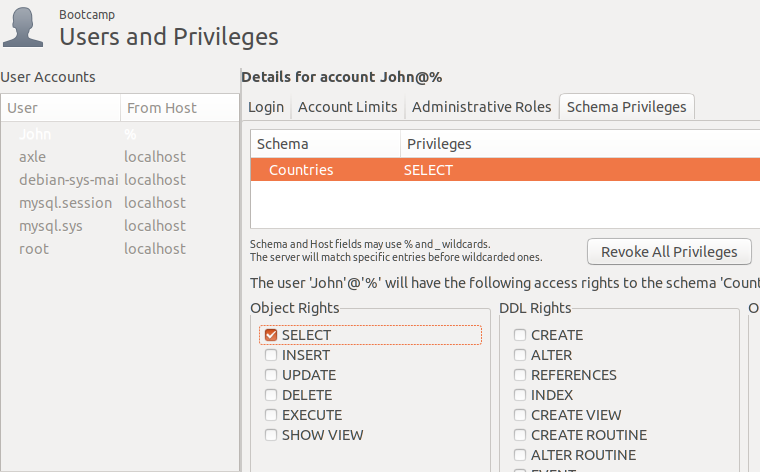
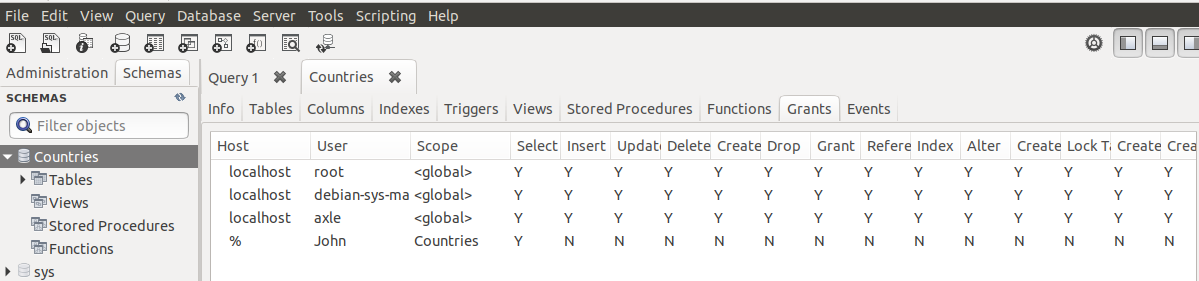
Note: At the time of developing this document there was a major disaster in Türkiye. I extend my heartfelt condolences and sympathies to everyone affected by this natural calamity.

1. Not only can we edit the records themselves, we can also edit certain meta-data such as the name of the column. Right click on the name of the table and choose Alter Table. You will see a lot of details about the table itself, this is meta data. Scroll down to GNP and try to change the value FLOAT(10,2) to FLOAT(15,2). Remember to click the Apply button. Close the tab and it may ask you to verify your changes one more time.
2. Verify the changes by closing that tab, then right click on country\_names table again, from the Schemas tab. Choose the Table Inspector option this time and click on the Columns tab. You should see the GNP changed there:  
     
   
3. So far we changed the data itself and the data type of a particular column. Lets try to change something about the database itself. From the menu bar, choose Server then Users and Privileges. Notice we have the option here to Add Account:
4. Click the Add Account button and add a new user by entering a name into the Login Name field. You can also give your user a password:

Remember to click on the Apply button to save the changes.

1. Now go to Schema Privileges tab and click on Add Entry. Click on the Selected schema radio button and then click the OK button at the bottom right of that dialog box:

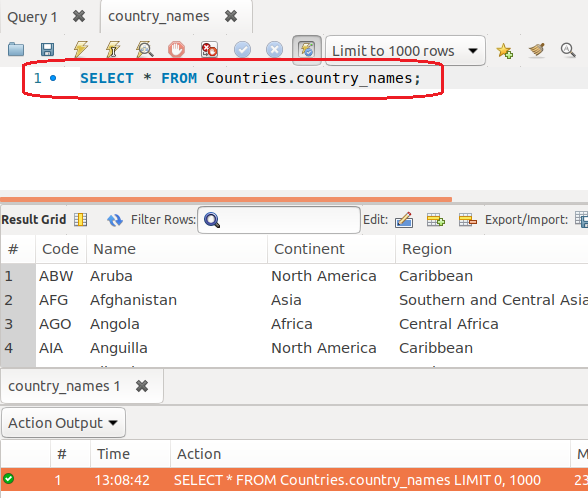


1. What we did was added a new user named *John*, we can now decide what privileges John should have on this schema. In my case here, John can only *select* data, he cannot *insert, edit* or delete any of the records:  
   Remember to click on Apply to confirm the changes.
2. If you wanted to confirm our actions, right click on Countries database and choose Schema Inspector. Once there go to the Grants tab you will see your user there along with all the privileges we gave to that user: 

|  |
| --- |
| End of section |

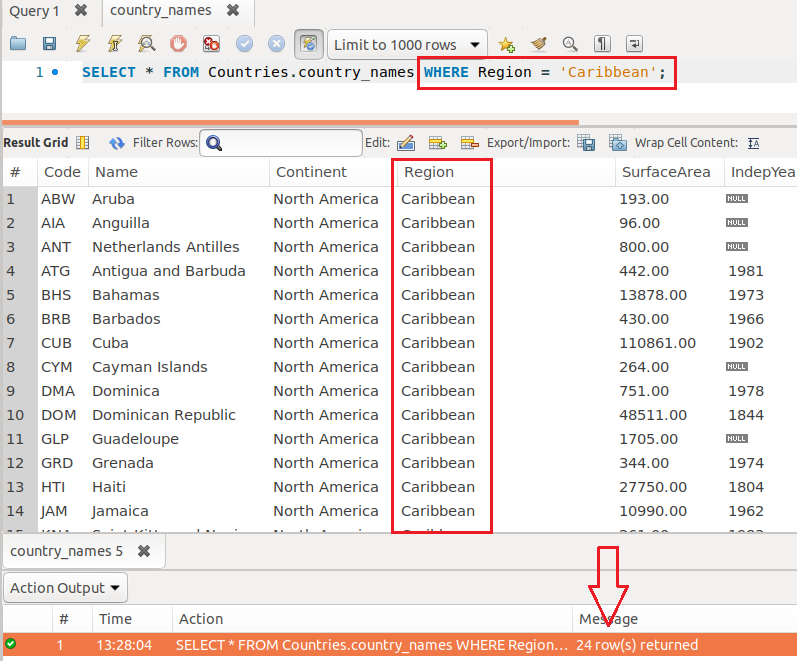
# Part 5 – Exploring SQL

In this section we will begin to use the statements that support some the UI actions you have performed so far. These statements are known as SQL commands or Structured Query Language. The Query part is what interfaces with the underlying data.

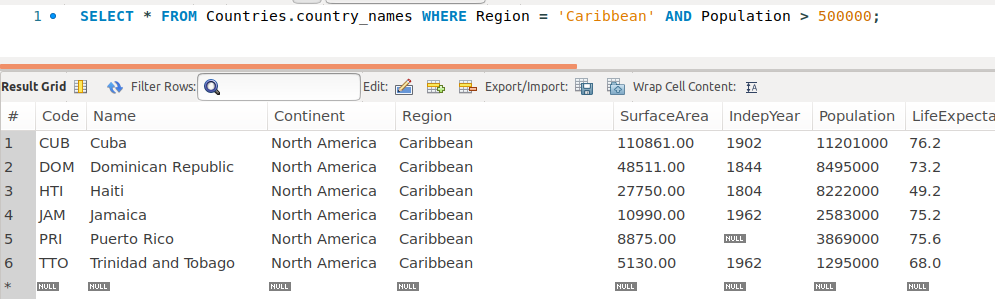
1. In Part 2 you right-clicked on the table and choose the option to *show rows*. Once that command is executed you had three windows in the main area of the UI.

The bottom window is the Action Output window, the middle area shows the Result Grid but the top window shows a line of code. That is the SQL statement that Workbench executed in order to get that middle window data.

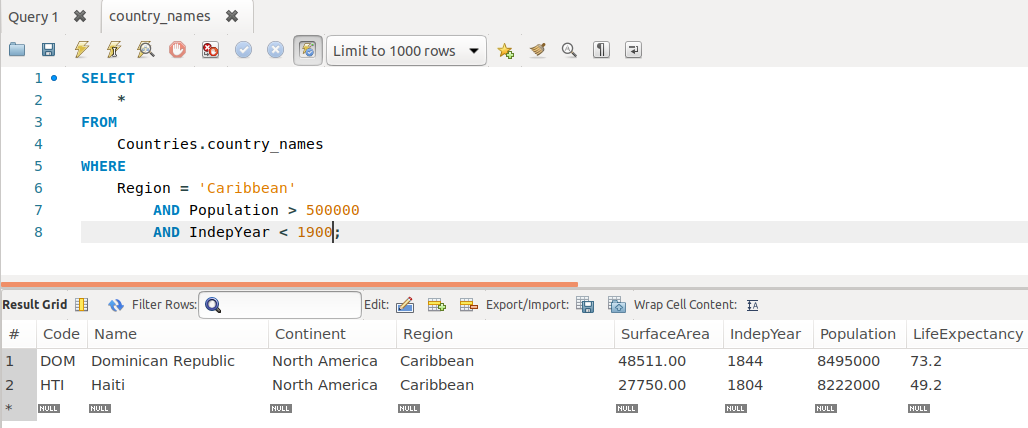
It instructed the database to show us all the data from that particular table. The countries that is in front of country\_names is considered good practice. It means that we got the data from this table and from the *Countries* database. There could be a different database that also has a country\_names table, so when written like this, there is no doubt where the data came from. This convention is sometimes called **Fully Qualified Names** and it is used in programming as well.

1. These SQL statements are very useful as it allows us to work directly with the database, tables and the data itself. For now we will concentrate on getting specific data. We can decide *how many* records we want as well as use *criteria* to return only those records that meet a specific criteria.

In the query above, notice the criteria. We asked only for those records that had the value of ‘Caribbean’ in the Region field. This query featured the **WHERE** keyword, an extremely important SQL keyword. In the Grid, only those records that matched the criteria we set, showed up, we say the *resultset* was returned. Also notice the red arrow, the Action Output window shows us exactly how many records were returned, 24.

1. With this information we can expand our query. What if we wanted all the countries in the Caribbean area, but also wanted to see only those countries from the Caribbean with population numbers over 500,000 people.

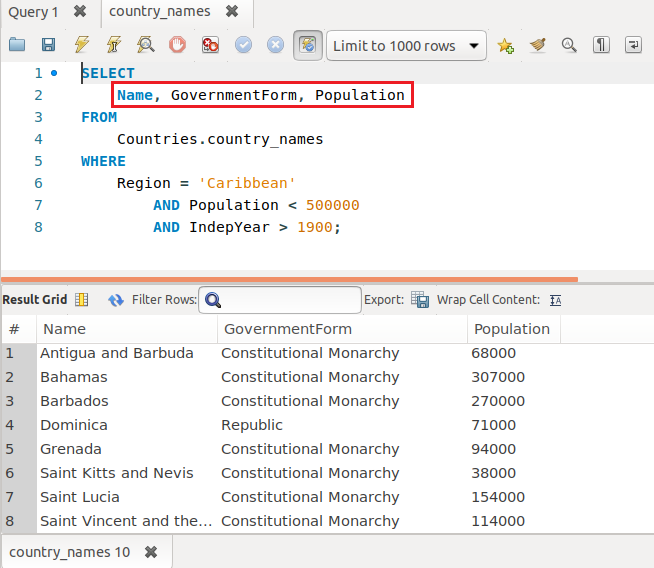
Notice that we include the **AND** keyword but also we can do some math. We asked that the values in the Population field, which happens to be numeric data, be compared against some value, 500,000 in this case.

1. Lets expand this query to return countries that match the previous criteria but gained independence before the twentieth century:

Notice that we had to use the **AND** keyword twice and only two countries showed up as a final result.

1. You may have noticed that the query has become large. There is a feature of Workbench that helps us structure our SQL code. You can find this feature under the main menu *Edit* then *Format* and finally choose *Beautify Query*:   
     
   Graphical user interface

   Description automatically generated
2. It is also possible to restrict the number of columns that return. This table has several columns and sometimes you only want to see certain columns:



|  |
| --- |
| Only 8 records were returned from this query. Notice that I changed the criteria a bit.  End of section |

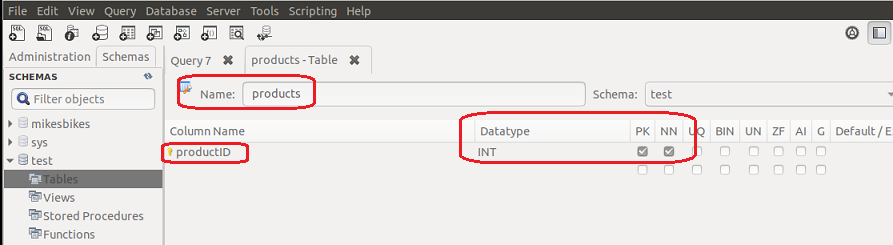
# Part 6 Introduction to Constraints

Databases exist because they add value to the business. One of the ways we add value is by restricting what can be done with our data. We can create rules that apply when working with tables and fields. You have already seen some of these rules, for example the *data type* of each column.

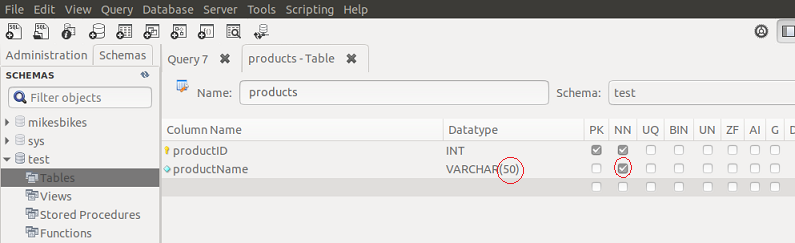
We wont allow an alphabetic character into a field designed to hold currency data. In this part we begin to explore even more *constraints*. Remember we do this in order to add safety to our data.

Another security measure is the *Primary Key*. This unique value identifies each row or record in a table. There may also be a *Foreign Key* in another table. This PK->FK connection is what gives rise to **relationships** amongst tables.

1. In Part1 #4 you created a new database. We will do the same now, just call your DB Test or something similar.
2. Now we create a table called products. Refer to Part1 #8. Right click on Tables directly under test. Notice that the table name is at the top, the IDE puts us at the first column. Lets call this column productID:



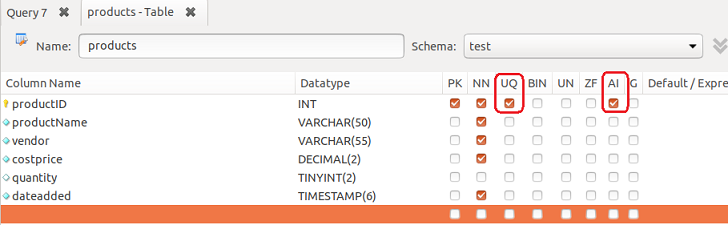
Notice that the *Datatype* was inserted automatically and the PK and NN fields are checked automatically.

1. Add another field called productName and make it a *varchar* with a maximum field width of 50 characters. Also check the **NN** field, this field cannot be empty:  
     
   
2. Add two more fields, one to store the vendor who supplies this product, the cost price of the product, the quantity in stock and the date the product was added to the store:  
     
   

Notice that the last two fields do not have any special additional constraints. We will fix that shortly. Also notice that costprice has a decimal type with space for just 2 decimal places. For quantity I chose *tinyint* since I don’t expect to have more than 99 items in stock of this particular item. The *dateadded* column is probably more interesting, it has the value of **TIMESTAMP(6)**.

This means that each time a record is added, the software wil insert the date/time that this happens. Nothing more for you to do as the programmer or administrator, it is done automatically for you. There are several features like this in most database systems.

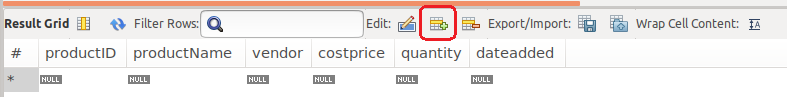
1. In terms of the PK, which is productID in this case, we have secured it very well. However remember this value must be unique in order to identify each record. So lets check the UQ box, which makes this field a unique one.
2. We can also make this field an auto increment field. This, just like the date will happen automatically as each record is entered. If no other value is inserted here, the system will start counting from 1 then keep adding 1 as we insert multiple records.

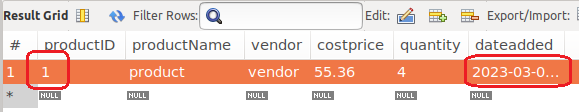


1. At this point, hit the Apply button at the bottom right of the UI and the system will respond with the SQL statement generated in order to accomplish all the tasks you have asked it to do:

|  |
| --- |
| CREATE TABLE `test`.`products` (  `productID` INT NOT NULL AUTO\_INCREMENT,  `productName` VARCHAR(50) NOT NULL,  `vendor` VARCHAR(55) NOT NULL,  `costprice` DECIMAL(4,2) NOT NULL,  `quantity` TINYINT(2) NULL,  `dateadded` TIMESTAMP(6) NOT NULL,  PRIMARY KEY (`productID`),  UNIQUE INDEX `productID\_UNIQUE` (`productID` ASC)); |

Note: this may not be exactly what we do in the future, but it is very close.

1. We can now add our first record to see how the table will respond. If you do not see the products table, remember to refresh the schemas. Right click on the table name and choose the first option “Select Rows…”
2. Use the UI to enter a new row, but start with productName, NOT productID. Remember productID, and dateadded are automatic. Alternatively you can just start entering data and remember to click the Apply button at the bottom right corner:  
   
3. If you now hit the Apply button on the bottom right of the UI, the entire record will be added:



Notice that the productID and the dateadded fields got values automatically.

1. (Optional) Add a second, third record. You will see the productID incrementing and the dateadded field automatically being populated. Try to enter a record with an alphabetic character in the quantity field. If you try to apply this record, it should reject since that field only takes numeric data. This is called a violation. To delete an entire row, just right click on that row and choose delete option. You will need to position the mouse in the # column in order to select the entire row.

# Part 7 – Database & Table Creation

In this section we create a new database, add a table then add a record. We will change the table after the first install, then change a record after insert. We will mainly use SQL instead of the Workbench UI for some of these events. Remove all databases except the sys database.

1. Execute the following SQL in one of the query windows. It will create a new database called **KarlsBikes**;

|  |
| --- |
| CREATE SCHEMA `KarlsBikes` ; |

1. Now we need to put the focus on KarlsBikes and then execute the create table command:

|  |
| --- |
| USE KarlsBikes; |

1. Here comes the table creation code:

|  |
| --- |
| CREATE TABLE `products` (  `pID` int NOT NULL AUTO\_INCREMENT,  `pName` varchar(70) NOT NULL,  `pCategory` varchar(10) NOT NULL,  `pVendor` varchar(50) NOT NULL,  `pDescription` text NOT NULL,  `quantity` smallint NOT NULL,  `costPrice` decimal(8,2) NOT NULL,  `MSRP` decimal(8,2),  PRIMARY KEY (`pID`)  ); |

Notice that we already added a few constraints. Apart from the data type, we made the pID auto increment and we added that column as a Primary Key towards the end of the statement.

1. Before we insert a record, lets make the pID start at a 1000. This will give us four digits for our product id:

|  |
| --- |
| ALTER TABLE `KarlsBikes`.`products` AUTO\_INCREMENT = 1000; |

Note, this is one line, and it has to be done AFTER the table is created.

1. We do not want two products to have the exact same description, so lets fix that by add the UNIQIUE constraint to the pName field:

|  |
| --- |
| ALTER TABLE `KarlsBikes`.`products`  ADD UNIQUE INDEX `pName\_UNIQUE` (`pName` ASC); |

1. We should also change the costPrice and MSRP fields. These should be larger to accommodate future expansion of the store. This statement also shows how to perform multiple changes in one statement:

|  |
| --- |
| ALTER TABLE `KarlsBikes`.`products`  CHANGE COLUMN `costPrice` `costPrice` DECIMAL(10,2) NOT NULL ,  CHANGE COLUMN `MSRP` `MSRP` DECIMAL(10,2) NOT NULL ; |

1. Finally we can insert our first record:

|  |
| --- |
| INSERT INTO `products`  (pName, pCategory, pVendor, pDescription, quantity, costPrice, MSRP)  VALUES  ('Treckie Pedal Assist 250','EAssist','Quantom Bicycles','Features two-tone paint with chrome accents',4,471.10,692.00); |

Notice that pID is not in the statement, it will be added automatically. Also, select KarlsBikes before running the statement.

Here are all the SQL statement as shown in my query window:

|  |
| --- |
| CREATE SCHEMA `KarlsBikes` ;  --  USE KarlsBikes;  --  CREATE TABLE `products` (  `pID` int NOT NULL AUTO\_INCREMENT,  `pName` varchar(70) NOT NULL,  `pCategory` varchar(10) NOT NULL,  `pVendor` varchar(50) NOT NULL,  `pDescription` text NOT NULL,  `quantity` smallint NOT NULL,  `costPrice` decimal(8,2) NOT NULL,  `MSRP` decimal(8,2),  PRIMARY KEY (`pID`)  );  --  ALTER TABLE `KarlsBikes`.`products` AUTO\_INCREMENT = 1000;  --  ALTER TABLE `KarlsBikes`.`products`  ADD UNIQUE INDEX `pName\_UNIQUE` (`pName` ASC);  --  ALTER TABLE `KarlsBikes`.`products`  CHANGE COLUMN `costPrice` `costPrice` DECIMAL(10,2) NOT NULL ,  CHANGE COLUMN `MSRP` `MSRP` DECIMAL(10,2) NOT NULL ;  --  INSERT INTO `products`  (pName, pCategory, pVendor, pDescription, quantity, costPrice, MSRP)  VALUES  ('Treckie Pedal Assist 250','EAssist','Quantom Bicycles','Features two-tone paint with chrome accents',4,471.10,692.00); |

# Appendix A – Common Command Line Queries

In order to add a new user and promote that user to admin, you must first login as root. Once you login the following command will create a user called axle with a password as shown:

CREATE USER 'axle'@'localhost' IDENTIFIED BY 'p@ssword';

After that, run the following command to make sure axle is part of the users table:

select user, host from mysql.user;

Now we promote axle:

GRANT ALL PRIVILEGES ON \*.\* TO 'axle'@'localhost' WITH GRANT OPTION;

Finally

FLUSH PRIVILEGES;

When using the MySQL Workbench UI, and you attempt to cerate a new table, you will see several columns that help us constrict the data going into the fields. You already saw the Primary Key, Not Null, Auto Increment and Unique. Here is a list of the others:

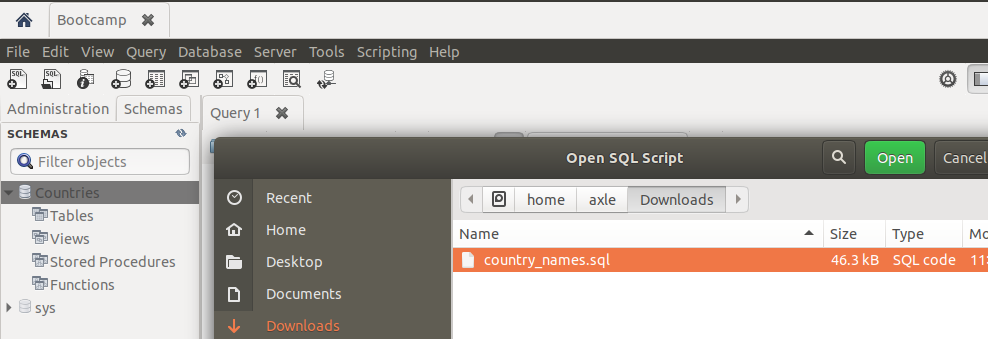
BIN - Binary (basically trie/false)

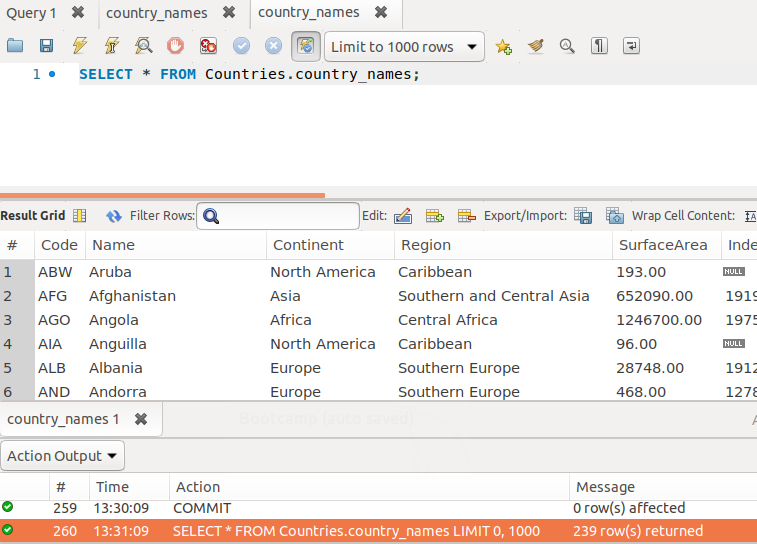
UN – Unsigned, the column will acceipt non-negative numbers only

ZF - Zero-Filled field (if the length is 5 like INT(5) then every field is filled with 5 zeros)

AI - Auto Increment

Importing using MySQL version 5:



1. We now need to execute all the insert lines, but there are several of them, one statement for each country in the world. One technique is to put your cursor on the first insert statement, line 31, then hold down the Shift key then CTRL-END. This will highlight all the lines. Leave them highlighted then click on the lightening icon without the I. So, this is the icon next to the floppy disk icon. Give it a moment to execute all the lines.
2. After this, if it goes will you should see some feedback in the Action Output window. If you see a gree checkmark, refresh as discussed in #5 above. Now drill down using the arrows next to Tables and right click on the country\_names table. Choose the Select Rows option and a third tab should now open showing all the countries you just imported into this table:  
     
   

You can now make changes to the content of these rows as we did in Part 2 #4.

1. Put your cursor on line 2, **USE Countries**. Once your cursor is on this line click on the  icon, it will execute that line until it encounters a semi-colon. Since you might have several databases on your computer, by executing this line, we tell Workbench to prepare this particular database for our use. In other words, Countries database will respond to any SQL statement we execute from this point forward.
2. Since we do not have a table named country\_names, we can skip this line and put the cursor on the line that creates a table with that name, so line 6. Once your cursor is on this line click on the  icon, it will execute that line until it encounters a semi-colon.

Graphical user interface, text, application

Description automatically generated

You will get a response in the Action Output pane below the main UI area. However if you click on the refresh button next to SCHEMAS on the left pane, you will see the arrow next to Tables. This indicates that we now have at least one table in this database.

# Appendix B – SQL Workbench Bug in Timestamp

Some of you may experience an issue when trying to insert a date/time field. As soon as you move the mouse, the Datatype field value changes back to something other than date/time. If this happens, try to change the field and wait until you get the Apply SQL Script to Database window. Once you have that window up, change the SQL manually:

ALTER TABLE `countries`.`test`

ADD COLUMN `dateadded` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP;

If the script is successful, close the query window and then re-open it.