Lab: Unit 02 – The Relational Model

# Overview

In this lab we will explore how the relational database model is applied in practice using the Microsoft SQL Server database management system.

## Learning Objectives

Upon completion of the lab, you should be able to:

* Explain the components of the relational model,
* Apply various components of the relational model in practice,
* Troubleshoot data integrity constraints, and
* Demonstrate how to create tables, keys, and constraints.

## What you will need

To complete this lab, you will need the Learn Databases Environment up and running, specifically:

* Microsoft SQL Server DBMS,
* The database provisioner application on <http://localhost:5000> to provision the **demo** database,
* The Adminer database administration tool, <http://localhost:5002> , login and connect to the **demo** database on your DBMS,
* Please review the first lab if you require assistance with these tools.

## The Database: Moze.com

Moze.com connects people who want their lawn grass cut (customers) with those who are willing to cut it (contractors). Here is the business model:

1. Customers submit a lawn cutting job for a specific date and time.
2. Contractors with billing rates within the customer’s min/max threshold and within the same city are matched and choose to accept the job.
3. Customer is notified when contractor accepts the job and is provided with a date and time of the service.
4. Once the job is done the customer can rate the contractor 1 to 5 stars.

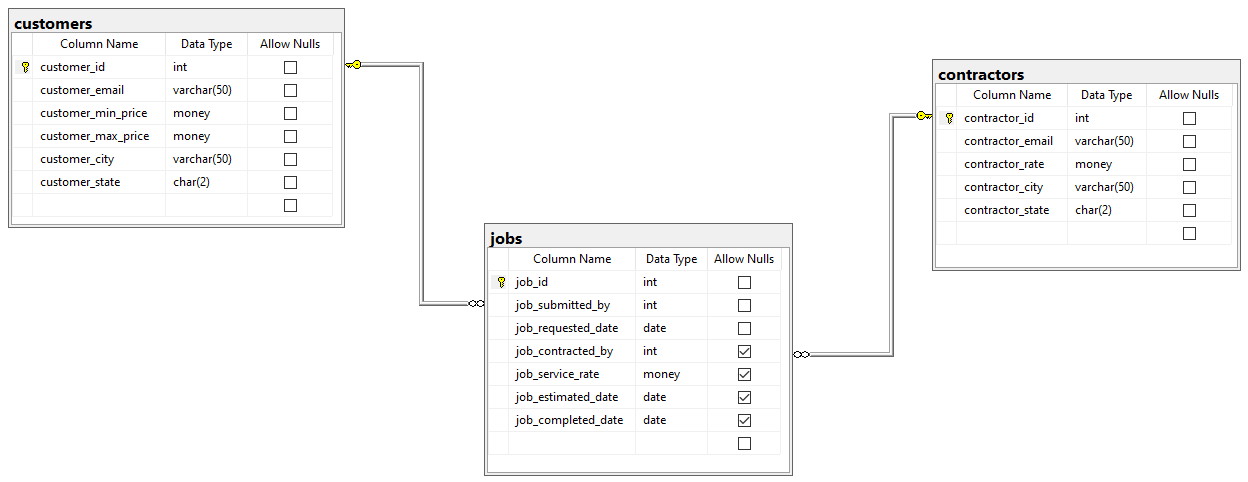
Here is the internal model of Moze.com. You will need to refer to this information throughout the lab.  


Figure 1Internal Model of moze.com

Table: **customers**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Domain** | **Comments** |
| customer\_id | Surrogate key not null | Will be the primary key |
| customer\_email | Varchar length 50 not null | Natural key |
| customer\_min\_price | Money not null | Smallest price the customer is pay for a lawn cutting |
| customer\_max\_price | Money not null | Maximum price the customer is willing to pay for a lawn cutting |
| customer\_city | Varchar 50 not null | US city of the customer |
| customer\_state | Char 2 not null | US state code of the customer (lookup table) |
|  | | |
| **Constraint Name** | **Type** | **Comments** |
| pk\_customers\_customer\_email | Primary key on customer\_id | Enforce pk over surrogate key on table |
| u\_customers\_customer\_email | Unique on customer\_email | Enforces natural key to establish entity integrity |
| ck\_customers\_valid\_prices | Check customer\_min\_price <=customer\_max\_price | Make sure the minimum price is below or equal tothe maximum price |
| fk\_customers\_customer\_state | Foreign key references state\_lookup table | A lookup table to restrict customer state to valid US states only |

Table: **contractor**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Domain** | **Comments** |
| contractor\_id | Surrogate key not null | Will be the primary key |
| contractor\_email | Varchar length 50 not null | Natural key |
| contractor\_rate | Money not null | The amount the contractor charges for the job |
| contractor\_city | Varchar 50 not null | US city of the contractor |
| contractor\_state | Char 2 not null | US state code of the contractor (lookup table) |
|  | | |
| **Constraint Name** | **Type** | **Comments** |
| pk\_contractors\_contractor\_email | Primary key on contractor\_id | Enforce pk over surrogate key on table |
| u\_contractors\_contractor\_email | Unique on contractor\_email | Enforces natural key to establish entity integrity |
| fk\_contractor \_contractor\_state | Foreign key references state\_lookup table | A lookup table to restrict contractor state to valid US states only |

Table: **jobs**  
The jobs table connects customer to contractor.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Domain** | **Comments** |
| job\_id | Surrogate key not null | Will be the primary key |
| job\_submitted\_by | Int not null | The customer id of the customer who submitted the job |
| job\_requested\_date | Date not null | When the job should be done |
| job\_contracted\_by | Int null | The contractor id of the contractor who accepted the job |
| job\_service\_rate | Money null | The amount of the job |
| job\_estimated\_date | Date null | The estimated date of when the job will be done |
| job completed\_date | Date null | The actual date the job was done |
| job\_customer\_rating | Int null | A integer value between 1 and 5 |
|  | | |
| **Constraint Name** | **Type** | **Comments** |
| pk\_jobs\_job\_id | Primary key on job\_id | Enforce pk over surrogate key on table |
| fk\_jobs\_job\_submitted\_by | Foreign key references customers table pk | Customer who submitted the job |
| fk\_jobs\_contracted\_by | Foreign key references contractors table pk | Contractor who accepted the job |
| ck\_valid\_job\_dates | Check requested\_date=<=estimated\_date and estimated\_date<=completed\_date | Make sure the dates are valid. |

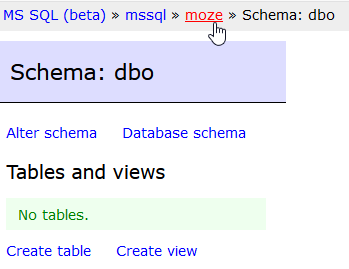
# Walkthrough

Since we’ve yet to learn SQL, we will use Adminer to create the tables and constraints. Please know that certain features like check constraints are not implemented in the Adminer so we will skip those for now.

## Let’s make the moze database.

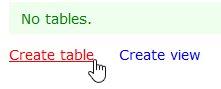
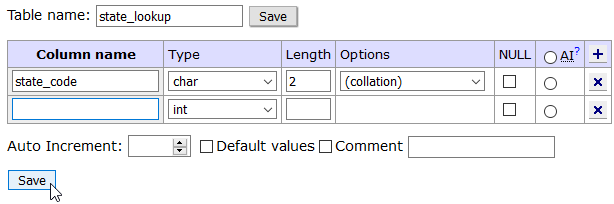
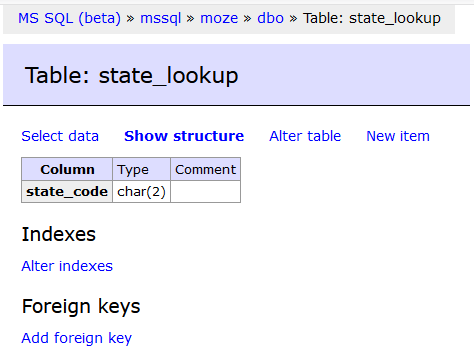
Database management systems support several databases on a single server instance. It makes sense to group your data and metadata into separate database for each application.

First, let’s create the **moze** database. From Adminier:

1. Click on the **mssql** link at the top 
2. This will display a list of all databases. Click **Create database**.
3. Enter **moze** as the database name and click **Save.**
4. When the database is created, Adminer will switch **moze** to the current database.   
   

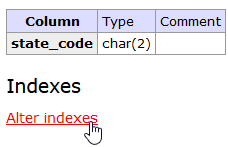
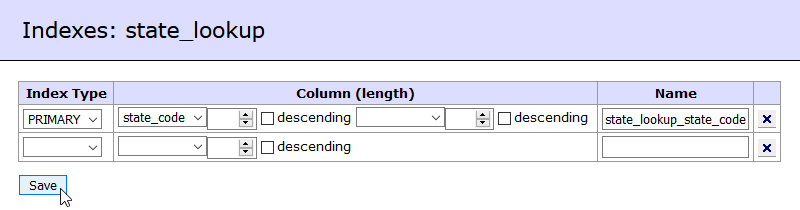
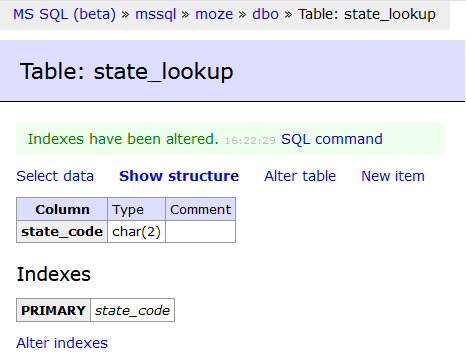
## Creating the state\_lookup table

Now that the database is created, let’s start by creating the state lookup table. As you may recall tables are metadata definitions which explain to the DBMS how the data are structured and which data domain rules must be followed for each column in the table. From Adminer:

1. Click on **Create table**   
   
2. Now you will see the Create table screen.
   1. For the Table name, enter **state\_lookup**
   2. Under Column name add **state\_code** of type **char** and length **2.**
   3. When you are finished click **Save.**
3. Once the table saves, you are taken to the table details screen for the **state\_lookup** table:  
   

## Creating a primary key

From the table details screen, we can add table-based constraints such as primary foreign keys. In this step we will set the primary key on the table.

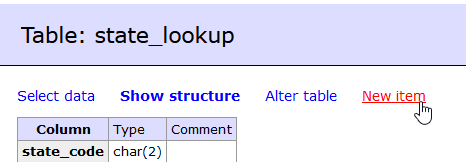
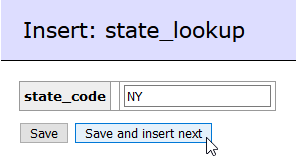
1. From this screen, Click on **Alter indexes.**
2. This will navigate you to the indexes screen for the **state\_lookup** table. Complete the index as follows:
   1. Index type: **PRIMARY**
   2. Column **state\_code**
   3. When your screen looks like this screen shot,   
        
      click **Save** toadd the primary key to the table.
3. Once again we return to the table details screen where we now see the index was added:  
   

## Adding Data

Now that the metadata has been defined in our database, let’s add some data. In this initial phase of the Moze.com app, they only serve the following states:

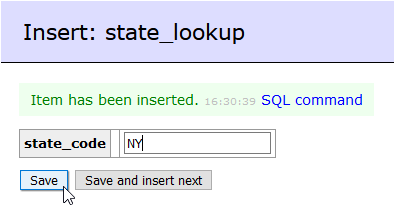
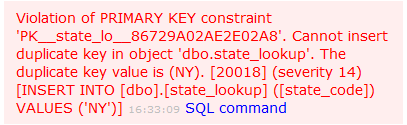
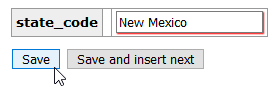
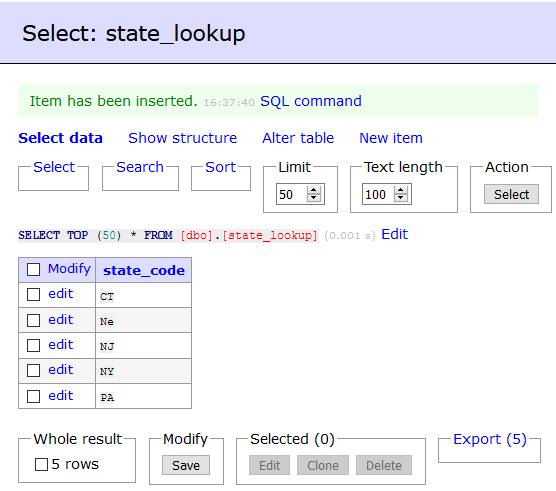
NY, PA, NJ, CT

Let’s add them to the **state\_lookup** table.

1. From the table details screen of the **state\_lookup** table, click on **New Item**
2. You will see a text box where you can enter a data for the **state\_code** type **NY**   
   
3. click **Save and insert next** when ready to add another state.
4. Repeat this process adding **PA**, **NJ** and **CT**.

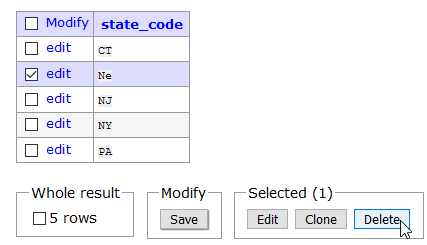
## Entity Integrity in Action

Let’s observe how the primary key establishes entity integrity by now allowing us to insert **NY** twice!

1. From the Insert: state\_lookup screen,   
     
   type **NY** and click **Save**.
2. You should see an error:  
     
   This error tells us that we cannot insert ‘NY’ into the database because doing so would violate the PRIMARY KEY constraint. This is exactly what we wanted! This is also one of the many aspects that makes the DBMS so popular it is easy to create data integrity rules which help to keep our data as accurate as possible!
3. Let’s try something else. This time type **New Mexico** in the state\_code text box and   
     
   click **Save**
4. You will return to the **Select** data screen, where you will see the 5 states entered.   
   
5. Huh? Why did it save **New Mexico** as **Ne**? You may recall that the data type of **state\_code** was char(2). If you provide more characters than what are allowed the DBMS simply truncates the data to length. Not what we wanted, but important to know!

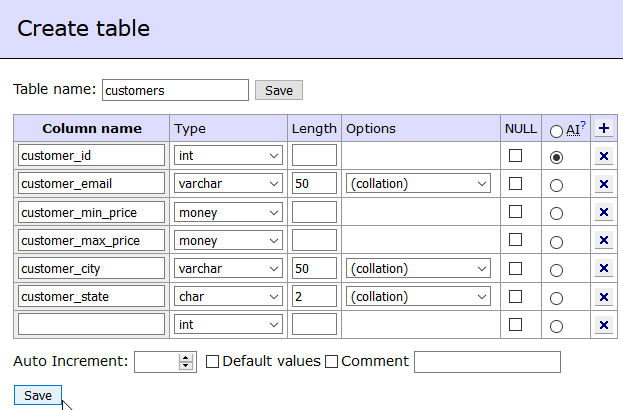
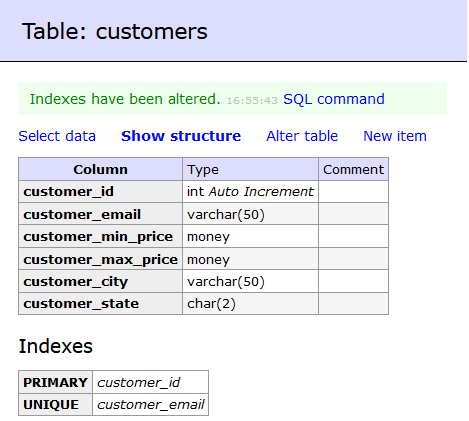
## Deleting Data

Let’s remove our mistake from the previous step. From the **Select data** screen:

1. Check the box next to **Ne** then click **Delete**
2. After you confirm the operation the row is removed. It is important to note that there is no practical way to undo this. Changes made to the data in a DBMS are persistent.

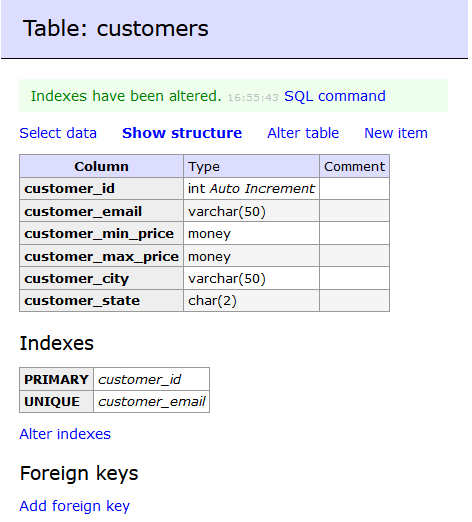
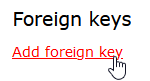
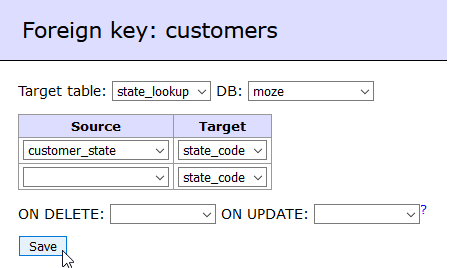
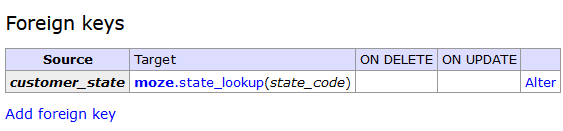
## Creating the customers table

Next, try to make the customers table on your own with little guidance. This table is a little more complex but should be straightforward. We won’t bother to walk you through the process so you’ll have to figure it out. The table definition appears in the overview section at the top of the page, so follow the details therein.

1. First create the table and columns, like so. Make sure to check the **AI** radio button for the customer\_id column so that it auto increments.   
   
2. If you screw up drop the table and start over. You CANNOT alter the table and set an auto-increment in Adminer.
3. After you create the table successfully, add the primary key and unique constraints for the natural key.
4. When you are finished this is how the **Show structure** screen should look for this table:  
   

## Adding a Foreign Key constraint: Referential Integrity

In this last part of the walkthrough we will add a foreign key on the **customer\_state** column in the **customers** tableso that its values must come from the **state\_lookup** table. This is how we will guarantee that a customer or contractor must come from one of these states: NY, NJ, PA, CT. This is a concept known as referential integrity and it is crucial to maintaining consistent data across business rules in a relational database. Here is how we do it in Adminer:

1. Bring up the **Show structure** screen for the **customers** table:  
   
2. Click on **Add foreign key** to add a foreign key to this table.   
     
   This means we will constrain the values over a column within this table to a set of primary key values in some other table.
3. You will now see the Foreign key screen.
   1. For Target table, choose **state\_lookup** this is the table with the primary key, also known as the referring table
   2. For Source, select the **customer\_state** column. This is the foreign key column.
   3. For Target, select the **state\_code** column. This is the primary key of the referring table.  
      
4. When ready, click **Save** to alter the table and add the foreign key. You will now return to the Show structure screen where you can see the foreign key:  
   

# Questions

Answer these questions using the problem set submission template. For any screenshots provided, please follow the guidelines for submitting a screenshot.

1. Does a table consist of data or metadata? Explain.

**A table consists of data and metadata both.**

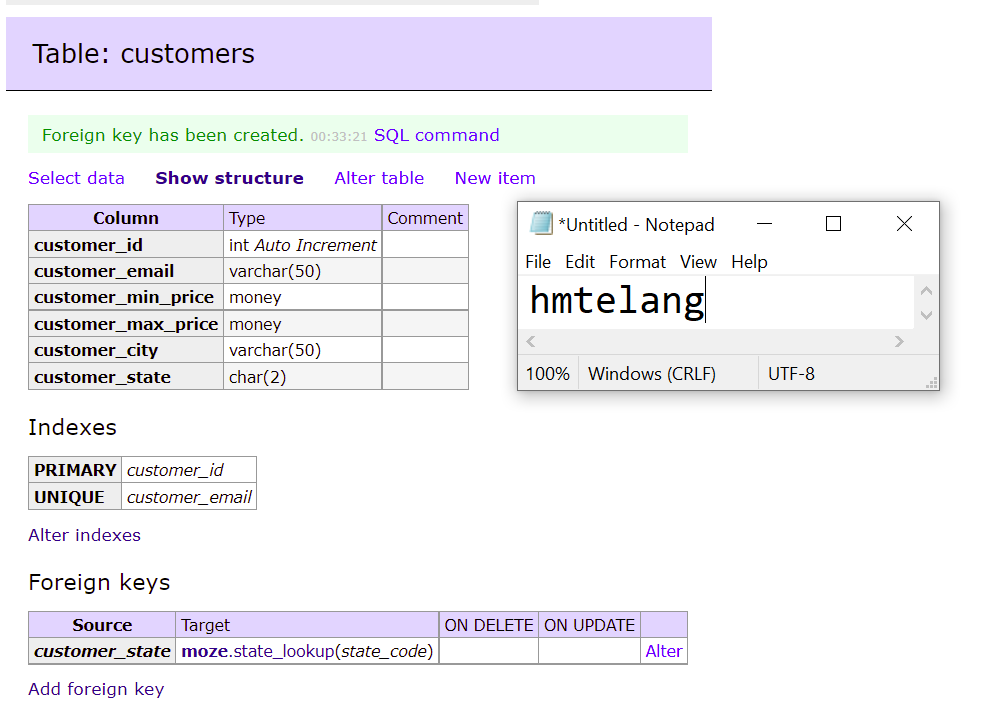
1. Describe what happens when you attempt to insert 200 characters into a column with a data type of varchar (50)?

**It will only insert the first 50 characters of the 200 characters inserted into the column.**

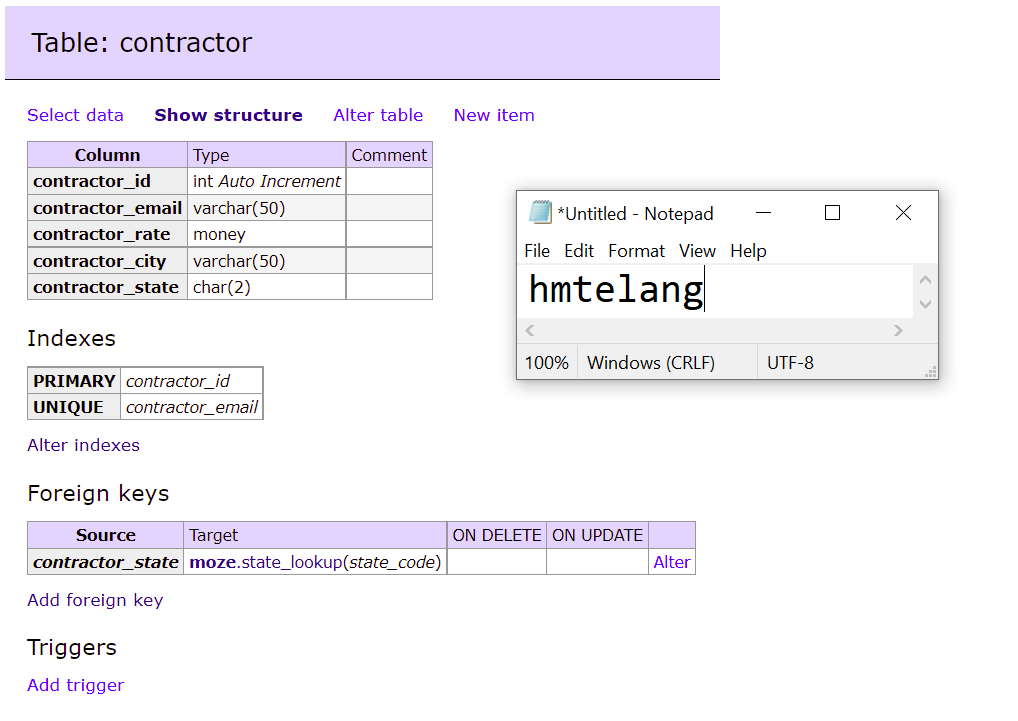
1. How do we enforce entity integrity over a table which uses a surrogate primary key?

**We can enforce entity integrity by using a composite primary key that includes the surrogate key and either a natural key such as an SSN or a VIN, or a secondary key that could provide some unique identification for example: a Department ID or an Employee ID.**

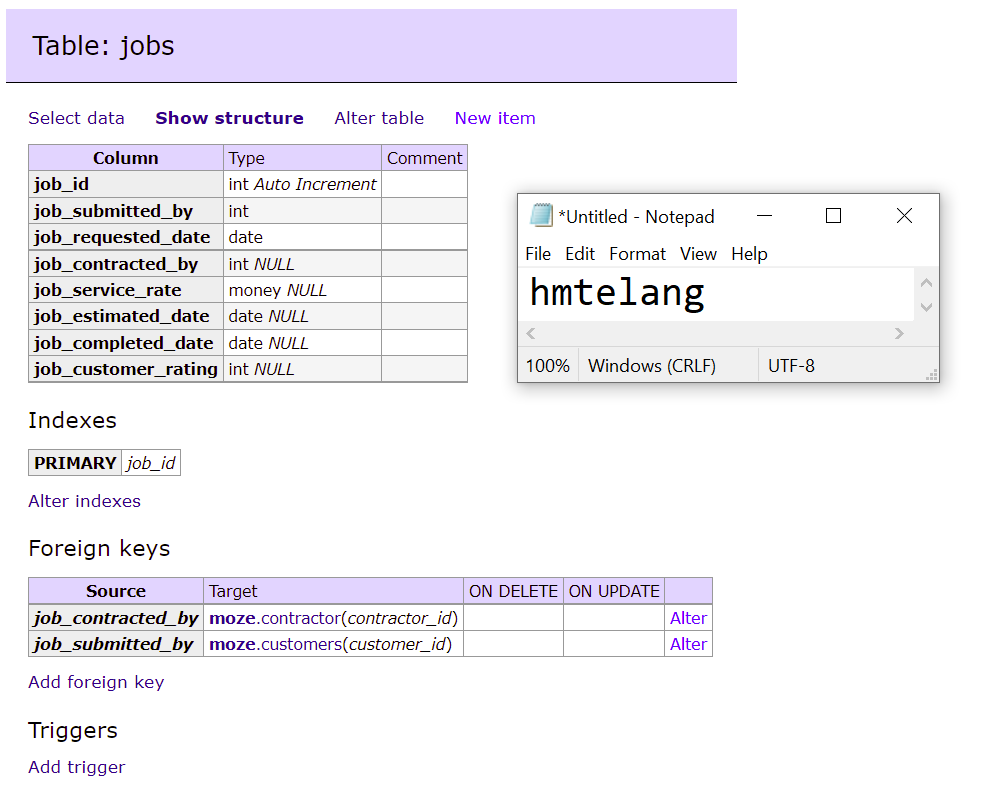
1. Provide a screenshot of your completed **customers** table include columns, indexes and foreign keys.



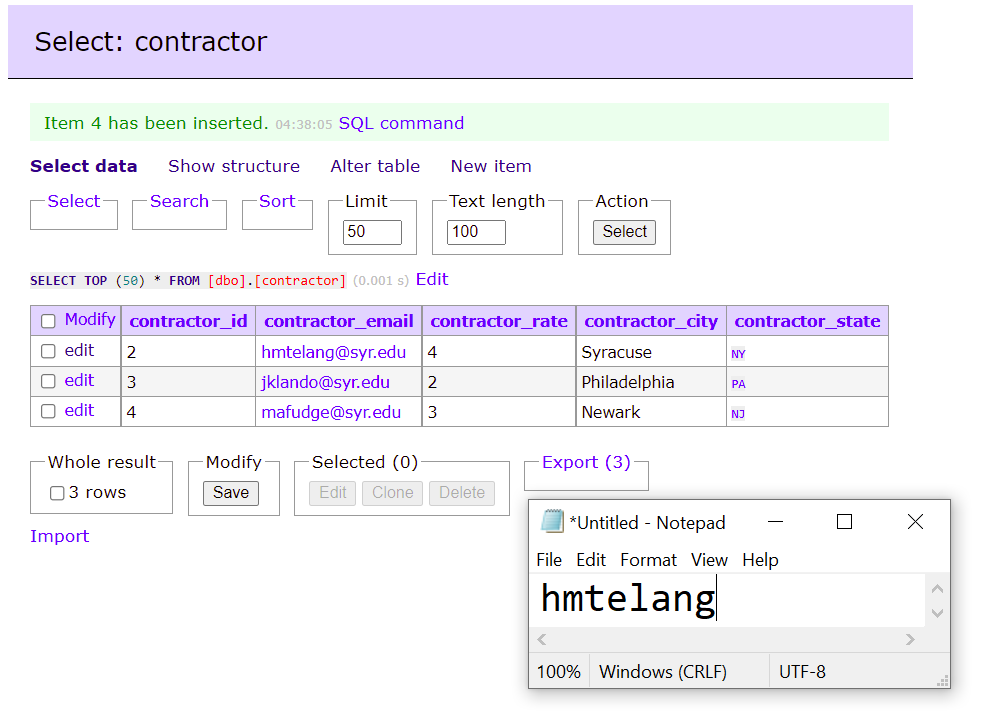
1. Implement the **contractors** table as defined in the overview section. Include columns, indexes (pk/unique) and foreign keys. Provide a screenshot of the table structure screen in Adminer and include the columns, indexes, and foreign keys sections.



1. Implement the **jobs** table as defined in the overview section. Include columns, indexes (pk/unique) and foreign keys. Provide a screenshot of the table structure screen in Adminer and include the columns, indexes, and foreign keys sections.



1. Add 3 contractors to the **contractors** table and provide a screenshot of the Select data screen as evidence they were added.



1. Can you add two contractors with the same email address? Explain.

**No. It is not possible to add two contractors with the same email address since a unique constraint has already been applied on the metadata ‘contractor\_email’ to ensure data integrity in the table.**

1. Can you add a contractor from the state of MA? Explain.

**No. We cannot add a contractor from MA because this state does not exist in the lookup table of ‘state\_lookup’ thereby violating the referential integrity of the database.**