Eddie Rossi (ejr8yc), Kevin Wan (kw2ae), Farid Nia (fr2md), Joshua Arabit (ja7ad), Jeff Li (jl9qc)

CS 4750: Databases

**Milestone 4**

1. Requirements
   1. Finish coding
   2. Test your application - write down what you do
   3. Copy your SQL commands out of your app and put them in your docs, with example result sets
2. Finish Coding
3. Test Application
   1. Test Plan
      1. When testing our database project we divided the testing into smaller testing modules so that we are able to have more control of what we were inputting and what we are expecting from the system. For our project we had to make sure that three major components were working; the database itself, the database interface, and the business logic that is incorporated into the project. With these major components in mind, we can create smaller tests for each module in order to test the full scope of the system. Lastly, once the systems has been tested properly in each of the modules we can then run a beta test on the system where we can act out how a user will be able to log into the system and conduct their business as a role in our system. One example of this beta test run through would be logging in as a “manager” of the system and being able to edit tables in the system but not being able to add or drop tables due to the fact that those permissions are only reserved for “administrators” of the system. If we are able to perform that actions correctly for the given role then we can deem that the system is working properly as a whole.
   2. Testing Database
      1. The first module for testing would be our database itself. During this portion of testing we had to look out for some key things; the schema of our database, the type of data being held in our tables, and how the data is being used between the tables. One of the tests that was already done for our database was to prove that it was in 3NF in the previous milestone. Proving that our database is in 3NF shows that our database has three important attributes; the database has lossless-join, the database always preserves our dependencies, and although we might have some extra data, our relationship between attributes will be kept throughout the database. Another test that will be done to the database is to make sure that each of the attributes in our tables have the correct type of data, such as an “orderID” being an “INT” or a “customerName” being a “CHAR” in order to model real life and to make sure that we constrain the data to be a certain type so that business logic can be done on the table if needed. The last thing to test is that our database models all the correct relationships and entity sets that were laid out during the creation of the ER diagrams. We can go through our ER diagram and make sure that we decompose the entity sets and relationships correctly by following rules for cardinality and relationship as well as multivariable attributes for entity sets.
   3. Testing Database Interfacing
      1. The second module for testing is our database interfacing web application. The database interface application is our web application that will let users interact with our database through the use of a graphic user interface in a browser. The web application also adds on some security for our database, because the application acts a buffer in between the user and the database to ensure that the inputs being provided by the user are correct and that only certain permissions are given to certain users. The first test for our database interface application is to make sure that we are able to talk to our database in phpMyAdmin through the apache web server that holds our application. We can test this by adding the correct files and credentials for database authentication and attempting to insert a query into one of the tables. For our project, we decided to test adding a new customer into the database. First we created a html form that will allow us to input a new customer information and a button to submit the form and execute the php script with the arguments given in the form. When the php script executes, the file that runs our database authentication file and lets us insert a new customer whenever the php script runs. Once we submit the form, we can check our database in phpMyAdmin to see the new entry and if we are able to see the entry then we know that we are interfacing correctly with our database and can continue with creating our forms with the proper authentication file.
   4. Testing SQL commands and logic
      1. After making sure that we are able to talk to our database through the use of the apache web server we can test that we are authenticating the user as a certain role with the database. We can test this by first running the grant command on the types of users in our database and give them the certain permissions that are needed for our project. For example we can only give schema changing permissions to the “administrator” user where they can add new tables or add columns to tables. After granting commands, we can then create a form that will take in the users credentials and have a user selector that will check if the user can be authenticated as a certain role. Once authenticated, we can test this by logging into the application as an admin and submitting a form that is created for adding a new table. When the form is submitted, then user should be authenticated as an admin and the php script will execute that will create a new test table into the database. We can check that the authentication work by checking the database in phpMyAdmin and checking if the new table exist in the database. On the other hand we can authenticate as another user role that is not an administrator and trying to add a table into the database. When logged in as a customer adding to the table should give us a permission denied response from the database due to trying to authenticate through the wrong credentials to change the schema of the database.
4. Copy SQL commands out of app into docs with example result sets

**SQL Query: → query for when a customer wants to sort products by a manufacturer**

SELECT ProdType, ProdID, ProdName, ProdPrice

FROM Product NATURAL JOIN Makes NATURAL JOIN Manufacturer

WHERE ManuName is ‘Nike’

**Expected output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ProdType** | **ProdID** | **ProdName** | **ProdPrice** |
| Stapler | 555563 | SuperStapler | 85.00 |
| Rock | 555575 | BigRock | 73.25 |

**SQL Query: → query for when a customer wants to see their order history**

SELECT Shipping Cost, OrderID, TotalTax, CustLName, CustFName

FROM Customer NATURAL JOIN Makes NATURAL JOIN Orders

WHERE CustID = \_\_\_\_\_

**Expected output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Shipping Cost** | **OrderID** | **TotalTax** | **CustFName** | **CustLName** |
| 15.00 | 222263 | 103.85 | Tony | Bennett |
| 7.99 | 222212 | 22.37 | Bruce | Willis |

**SQL Query: → query for when a customer wants to see their return history**

SELECT RefundAmt, ReturnDate, ReturnQuantity, ProdName, ProdPrice

FROM Customer NATURAL JOIN Refunds NATURAL JOIN Returns NATURAL JOIN Has NATURAL JOIN LineItem NATURAL JOIN Product

WHERE CustID = \_\_\_\_\_

**Expected output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RefundAmt** | **ReturnDate** | **ReturnQuantity** | **ProdName** | **ProdPrice** |
| 77.00 | 12/03/2019 | 2 | SuperStapler | 38.50 |
| 125.37 | 1/1/2019 | 1 | MegaStapler | 125.37 |

**SQL Query: → query for when a manager wants to see the location of the majority of customers to design better marketing plan**

SELECT Count(CustID) as NumCustomers , State

FROM Customer NATURAL JOIN LocatedIn NATURAL Location

GroupBy State

SORT BY Count(CustID), Desc

**Expected output:**

|  |  |
| --- | --- |
| **NumCustomers** | **State** |
| 1250 | NY |
| 750 | VA |

**SQL Query: → query for when a manager wants to see the most sold items**

SELECT SUM(LineItemQty) as QtySold, ProdName

FROM Orders NATURAL JOIN LineItem NATURAL JOIN Product

GROUP BY ProdName

SORT BY QtySold Desc

**Expected output:**

|  |  |
| --- | --- |
| **QtySold** | **ProdName** |
| 18652 | SuperStapler |
| 1 | MegaStapler |

**SQL Query: → query for when a manager wants to find the customers of the year with the most number of orders**

SELECT CustID, CustFName, CustLName, COUNT(OrderID) as NumOrders

FROM Customer NATURAL JOIN Makes NATURAL JOIN Orders

GROUP BY CustID

SORT BY COUNT(OrderID) desc

**Expected output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CustID** | **CustFName** | **CustLName** | **NumOrders** |
| 999942 | Phony | Bennett | 18652 |
| 999943 | Michael | Strahan | 1 |