# CSCI 5559 - Database Systems
# Spring 2015
# Assignment 1 - Part b

**Due: 03/11/2015 8:59pm**

## Objectives

1) You are provided with schema and instance for a database (which is similar to the one you designed in Part ‘a’ of Assignment 1. You are asked to create and populate tables/relations according to this schema using Microsoft SQL Server, a popular commercial DBMS (**10 points**).
2) You are provided with the template of a C# application that uses the aforementioned database as the backend. The only missing pieces of the application are the SQL queries that interact with the database. You are asked to design and implement these queries by populating and completing the provided C# application template (**90 points**).

## Preparations

1) Create an account on Microsoft DreamSpark using your CU Denver email address.

2) Download and install “Microsoft Visual Studio 2013 with Update 4” form Microsft DreamSpark. Use Google Chrome and make sure that pop-ups are allowed. Click here for guidelines.

3) Install .NET Frameworks 3.5 on your machine. Click here for guidelines.

4) Download and install “Microsoft SQL Server 2014 Enterprise Edition” from Microsoft DreamSpark. Click here for guidelines and important information.

5) Open Microsoft SQL Server, create and populate tables (from excel files) and write your queries. DO NOT CHANGE THE NAME OF THE COLUMNS AND TABLES. DO NOT ADD OR DELETE COLUMNS.

6) Open the application in “Microsoft Visual Studio” and complete the code. DO NOT MODIFY THE CODE EXEPT THE PARTS THAT YOU ARE ASKED TO COMPLETE. Click here for guidelines.

## Description

### Step 1: Database Creation (10 points)

Create and populate a database according to the database schema and data included in the enclosed Excel file, **Tables.xlsx,** where primary keys are underlined and foreign keys are shown in blue. You must use Microsoft SQL Server as the hosting DBMS. Your submission for this part of the assignment is the CREATE SQL scripts for each table containing the foreign keys and primary key constraints.

### Step 2: Queries (90 points total)

Notes:

- If a query does not work properly when embedded in the application but works fine directly on Microsoft SQL Server, you will receive %50 of the total point for that query.
- You can either include your queries **within the application** or create **stored procedures** in Microsoft SQL Server and call them from the application. The latter option would be easier in most cases.
- The data used in screenshots is different. Thus, your results would be different.

Assumptions:

- Each person can work in at most 1 store/branch.
- Each person has exactly one phone number.

#### Persons Tab Queries (10 points)

**1)** Write a query that receives three parameters @NAME, @FAMILY, and @EMAIL as input, and returns the persons (**excluding employees**) who satisfy the criteria indicated by the parameters. Please note that if, e.g., @NAME = "**or**" all persons that their first name include "**or**", such as *Ge**or**ge* and *Do**ro**thy*, should be included in the result. The same is true about the last name and email address. If all input parameters are set to empty strings, all persons should be returned as the result. Sort the results based on the last name ascending, first name ascending and person code ascending (**4 points**).

**Note: The list of the attributes to be returned as result of the query follows**
*PERSONCODE, ADDRESSCODE, FIRSTNAME, LASTNAME, EMAIL, DOB, STREET_NO, STREET_ADDRESS, CITY, [STATE], ZIP, X, Y, PHONE_NUMBER, PHONE_TYPE*

**Hint: Use the "*LIKE*" operator.**

**2)** Write a query which returns the name of the cities and the number of persons living in each city. Sort the results based on the number of persons descending and city names ascending (**2 points**).

| City | Persons |
|---|---|
| Denver | 9 |
| Castle Rock | 8 |
| Centennial | 8 |
| … | … |

**3)** Write a query that receives a city name as input parameter (@CITY) and returns the minimum and maximum date of birth of the persons living in that city (**4 points**).

| MIN DOB | MAX DOB |
|---|---|
| 01/01/1959 | 01/11/1999 |

## Items Tab Queries (18 points)

**4)** Write a query with 2 parameters (@NAME and @BRAND) which returns those items satisfying the criteria. Please note that, if the @Name = "**m1**" all items that their name include "**m1**", such as *ite**m1*** and *ite**m1**12*, should be returned. The same thing is true about the brand name. If all parameters are equal to empty strings, all items should be returned as the result. Sort the results based on item code ascending (**2 points**).

**Note: The list of the attributes to be returned as result of the query follows**
*ITEMCODE, NAME, BRAND, PRICE and [DESCRIPTION]*

**Hint: use *LIKE* operator.**

**5)** Write a query which displays store codes, item codes and the number of items sold in that store. Sort the result based on the store code (ascending) and the number of sold items (descending) and item code (ascending) (**3 points**).

| STORECODE | ITEMCODE | SOLD |
|---|---|---|
| 1 | 4 | 6 |
| 1 | 3 | 5 |
| 1 | 6 | 3 |
| 2 | 9 | 7 |
| … | .. | .. |

**6)** Write a query which returns store codes and item codes which are running out (less than 5 items left in stock). Sort the results based on the store code and remaining items and item code. (**3 points**).

| STORECODE | ITEMCODE | QUNATITY |
|---|---|---|
| 1 | 4 | 1 |
| 1 | 3 | 4 |
| 2 | 1 | 2 |
| 2 | 9 | 2 |
| … | … | … |

**7)** Write a query to find the best seller item. Query should display the item name and the total number sold. If there is more than 1 best seller item, your query should return only one of them with the smallest item code. For example, assuming that items 5 and 6 are sold 199 times and are best seller items, the expected result is shown in the following table (**5 points**).

**Hint: use *TOP* operator.**

| ITEMNAME | SOLD |
|---|---|
| Item 5 | 199 |

**8)** Write a query which returns the most popular brand name and the number of sold items of that brand. If there is more than one most popular brand, your query should only return the brand which comes first in the alphabetical order. For example, assuming that brands 1 and 2 are most popular brands with 499 sold items, the expected result is shown in the following table (**5 points**).
**Hint: use *TOP* operator.**

| BRAND | SOLD |
|---|---|
| Brand 1 | 499 |

## Transactions Tab Queries (8 points)

**9)** Write a query with 2 parameters (@LASTNAME and @PAYMENTMETHOD, which returns those transactions satisfying the criteria indicated by the parameters. Please note that, e.g., if the @LASTNAME = "**or**" all transactions that their buyers' last name include "**or"** should be selected. The same thing is true about the payment method. If all parameters are equal to empty strings, all transactions should be returned as the result. Sort the results based on Transaction code ascending (**2 points**).

**Note: The list of the attributes to be returned as result of the query follows**
*TRANSACTIONCODE, FIRSTNAME, LASTNAME, STORECODE, TRANSACTION_DATE, PAYMENT_METHOD*

**Hint: use *LIKE* operator.**

**10)** Write a query with the parameter (@TRASNACTIONCODE) which returns the items, their quantity and their price in the identified transaction. Sort the result based on the item name ascending. (**4 points**).

| NAME | QUANTITY | PRICE | TOTAL (Quantity × Price) |
|---|---|---|---|
| **Item 1** | 3 | 15 | 45 |
| **Item 4** | 1 | 60 | 60 |
| **Item 21** | 2 | 14 | 28 |

**11)** Write a query which returns the name of the cities and the number of transactions belonging to stores located in that city. Sort the result based on the number of transactions descending and city names ascending (**2 points**).

| City | Transactions |
|---|---|
| Denver | 90 |
| Castle Rock | 80 |
| Centennial | 80 |
| … | … |

## Map tab Queries (11 points)

**12)** Write a query that receives the coordinates of a location (i.e., @X and @Y) as input and returns the code for 5 stores nearest to the selected point as well as their Euclidean distances to the point (see below for the definition of Euclidean distance). The query should also return the x and y position of the store address (you could use X and Y columns for this). You have to use the **LOCATION** column to compute distance (not the X and Y columns). You do not need to define a spatial index on the **LOCATION** column. Sort the results based on the distance and store code (**5 points**).

**Hint: Use the *TOP* operator and the *STdistance* function.**

| Store | Distance | X | Y |
|---|---|---|---|
| 3 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 1 | 10 | 3 | 1 |
| 4 | 10 | 6 | 4 |
| 5 | 12 | 7 | 5 |

**13)** Write a query that receives coordinates of a location (i.e., @X and @Y) as well as a range @RANGE, and finds stores which are within the selected range centered at (X, Y). The query should return Euclidean distances from the center point (X, Y) as well. It should also return x and y position of the store address (you could use X and Y columns for this). You have to use the **LOCATION** column (not X and Y columns) in your query. You don't need to define any spatial index on the LOCATION column (**6 points**).

**Hint: Use the *TOP* operator and the *STdistance* function.**

| Store | Distance | X | Y |
|---|---|---|---|
| 3 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 1 | 10 | 3 | 1 |
| 4 | 10 | 6 | 4 |
| 5 | 12 | 7 | 5 |
| … | … | … | … |

## Other Queries (43 points)

**Write (implement) below queries directly in the database server**

**14)** Write a stored procedure that inserts a new person into the PERSON table. Parameters are @ADDRESSCODE, @NAME, @FAMILY, @DOB, and @EMAIL (**2 points**).

**15)** Write a stored procedure that receives a person's code (e.g. @PERSONCODE) and her new information, including name, family, date of birth, and email address) and updates the information of that person. (**2 points**).

**16)** Write a stored procedure that deletes the employee whose EMPLOYEECODE is equal to @EMPLOYEECODE. Your stored procedure should delete her records from WORKS_IN tables as well. (**3 points**).

**17)** We are interested in knowing which items are usually bought together. Write a stored procedure that receives the code of an item (i.e., @ITEMCODE) and returns the item code of another item that appears most frequently along with the first item in all transactions (**5 points)**.

**18)** Write a query with the parameter @STORECODE which returns the total sales amount for the selected store. (**5 points).**

| Total Sale |
|---|
| 13750 |

**19)** Write a query with the parameter @STORECODE which returns the total amount of weekly salary paid by the selected store (**6 points).**

| Weekly Salary |
|---|
| 3750 |

**20)** Write a query with the parameter @STORECODE which returns the date at which the store had its highest sales amount (**6 points).**

| Best Date |
|---|
| 01/01/2014 |

**21)** Write a query that returns the percentage of the persons who shopped at least once from a store located in a city other than the city in which they live **(3 points)**.

| Percentage |
|---|
| %23 |

**22)** Write a query that calculates the total price of all items in the inventory of all stores (note: not per each single store) **(5 points)**.

| Total Price |
|---|
| 999999 |

**23)** Write a stored procedure implemented directly in the database server which receives coordinates of four points (e.g. @X1, @Y1, @X2, @Y2, @X3, @Y3, @X4, @Y4) and returns all store codes that are inside the region identified by these 4 points in sequence. It should also return x and y position of the store address (you could use X and Y columns for this). You have to use the **LOCATION** column (not X and Y columns) in your query. You don't need to define any spatial index on the LOCATION column (**6 points**).
**Hint: Use the *STIntersects* function.**

| Store | X | Y |
|---|---|---|
| 3 | 1 | 1 |
| 2 | 2 | 2 |
| 1 | 3 | 1 |
| 4 | 6 | 4 |
| 5 | 7 | 5 |
| … | … | … |

## Preparation Guidelines/Tutorials

### Installing Visual Studio

You could find a tutorial here.
If your operating system is Windows 8, you do not need to use “Daemon Tools” or similar application to open the .ISO file. Windows 8 could do that for you. Just double click on the ISO file and click on the setup/install.exe.

### Installing .NET Frameworks 3.5

You could find a tutorial here.
Open the start menu and type “**feature**” in the search box. Choose “**Turn Windows features on or off**”. Find **.NET Framework 3.5** and check it (if it is unchecked). Windows will download and install it for you.

### Installing SQL Server

You could find a video tutorial here.

**Always choose default settings** (such as **default** in the **Server Configuration** step) and do not change the pre-selected options.
Make sure to choose **“Windows Authentication”** in the **Instance Configuration** step (**different than the tutorial**).

Make sure to choose **“Windows Authentication”** in the **Database Engine Configuration** step (**different than the tutorial**) and click on the **Add current user** button.

In the **feature selection step**, check all features under the **Instance features**. Make sure to check **Management tools (basic and advanced)**. You don’t need other features for this assignment.

### Opening the Application in Visual Studio

Unzip the project file and click on the “**A1.sln**”. Visual studio will open the project for you. In the solution browser, find the ***“DataLayer.cpp”*** and open it. It is the only file that you need to modify to complete your homework. Please **do not modify** other files. Under the **“Object Classes”** folder, you could find corresponding classes of tables. Under the **“ViewModelClasses”** you could find classes that will used for your query result. For example, **StoreCities.cpp** is a combination of Store and Address classes.

## How to Complete Queries

The connection method is already implemented for you. You only need to call ***Connect()*** method to get connected to the database and run your command. However, you need to modify the connection string and replace the database name with your own database name (if it is not “Assignemnt1”):

```
const string _ConnectionString = "Server=localhost;Database=Assignment1;Integrated
Security=SSPI;";
```

Here is a general form of code:

```
//this method is used for query no 23. The fields/properties that you need to
populate for each object is listed.
/*********Q23
storeLocation._STORECODE
storeLocation._Distance
storeLocation._X
storeLocation._Y
*******/
public static List<StoreDistance> FindStoresInRange(Point clickedPointX,
clickedPointY, int range)
        {
            List<StoreDistance> neareststores = new List<StoreDistance>();
            StoreDistance storeLocation;

            if (Connect())
            {
                //we are going to call a stored procedure called Q1
                using (SqlCommand sqlCommand = new SqlCommand("Q1", _Connection))
                {

                //setting the command type (StoredProcedure or Text (if you
                  directly write your query in the application)

sqlCommand.CommandType = CommandType.StoredProcedure;

//adding parameters to your sqlCommand here,

sqlCommand.Parameters.Add("@X", SqlDbType.VarChar).Value =
clickedPointX.ToString();
sqlCommand.Parameters.Add("@Y", SqlDbType.VarChar).Value =
clickedPointY.ToString();
//… add all parameters

                    SqlDataReader reader;

                    try { reader = sqlCommand.ExecuteReader(); }
                    catch (Exception ex) { return null; } // if sqlCommand failed
                                                           -> return null

                    //Converting query results to PersonPhoneAddress objects
                    while (reader.Read())
                    {
                        //creating a new instance of the object
                        storeLocation = new StoreDistance();
                        // populating it with query results
                        storeLocation._STORECODE = Convert.ToInt32(reader[0]);
                        storeLocation._Distance = …

                        //then adding the new object to the return list.
                        neareststores.Add(storeLocation);
                    }
                    reader.Close();

                    // returning the results
                    return neareststores;

                }
            }

            //Couldn't connect to the database -> return null
            else
                return null;

        }
```

If the return type contains another object inside it (for example, each instance of StoreCities.cpp class, contains Address and Store objects as its fields) you have to use **.** ***(dot)*** operator to access to their fields.
For example:
*storeCities = new StoreCities;*

*storeCities**.**Address**.**X = …;*

*StoreCities**.**Address**.**Y = …*

## How to Create a Stored Procedure in Microsoft SQL Server

You find the Microsoft tutorial on stored procedures here.

There is another tutorial here.

## Other Implementation Hints

- If the field type is int, double or float, you need to convert the reader value to an integer using the **Convert** method:

  ***Convert.ToInt32(****reader[0]**)**;*

- If the field type is string, you need to convert the reader value to string using the **ToStirng** method:

  *reader[1].**ToString()**;*

- If the field type is date, you need to cast the reader value to DataTime:

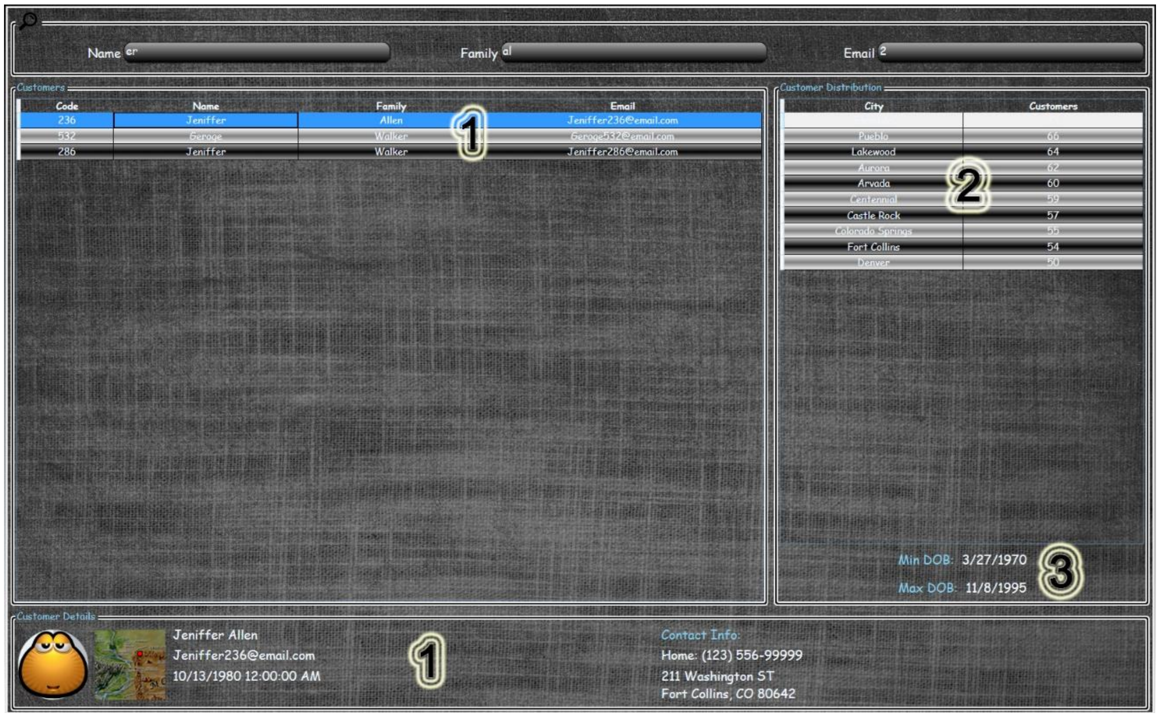
  ```
  (DateTime)reader[4];
  ```

- Make sure to put column/table names that are SQL Server keywords (e.g., "***STATE***" column in the "***ADDRESS***" table) inside brackets (e.g. ***[STATE]***) otherwise you will receive an error prompt.

- You are highly encouraged to directly write and test your queries on the database server first to make sure they are functional, and then complete the corresponding methods in the application accordingly.

## Screenshots

The locations where your query results must be displayed in the C# application GUI are shown in the following screenshots. If your queries are functional and your methods populate and return the return list/variable, the results will be automatically displayed on the GUI.



Name er    Family al    Email 2

Customers

| Code | Name | Family | Email |
|---|---|---|---|
| 236 | Jeniffer | Allen | Jeniffer236@email.com |
| 532 | Geroge | Walker | Geroge532@email.com |
| 286 | Jeniffer | Walker | Jeniffer286@email.com |

Customer Distribution

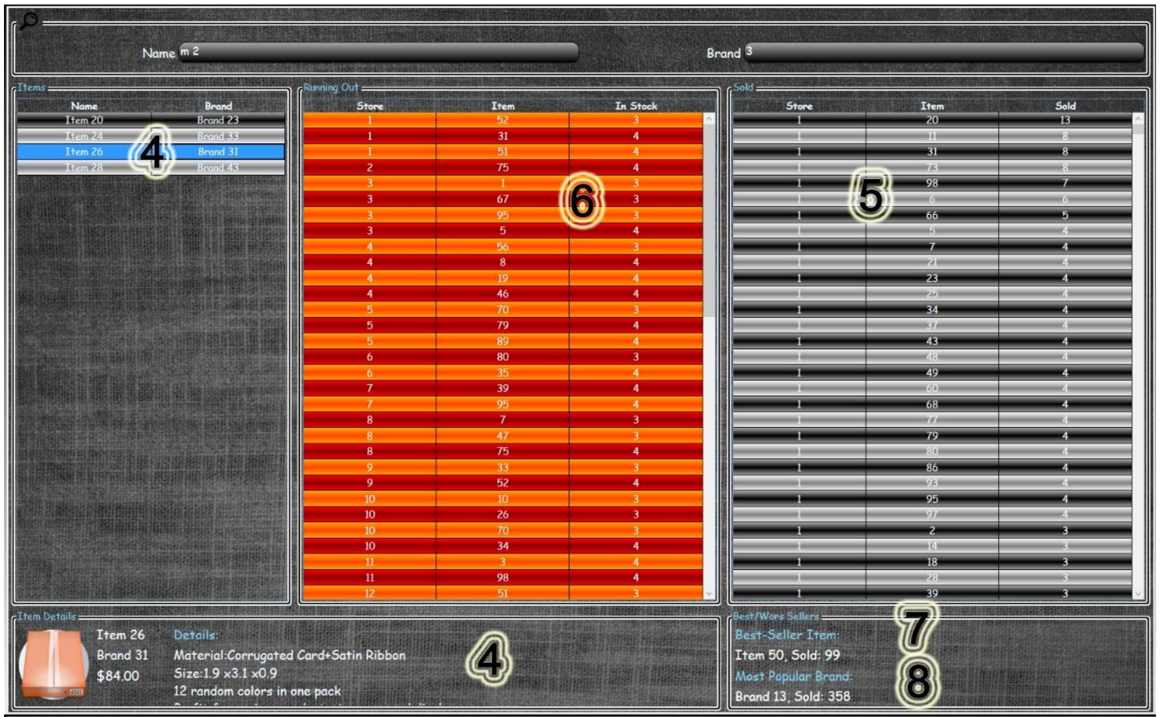
| City | Customers |
|---|---|
| | |
| Pueblo | 66 |
| Lakewood | 64 |
| Aurora | 62 |
| Arvada | 60 |
| Centennial | 59 |
| Castle Rock | 57 |
| Colorado Springs | 55 |
| Fort Collins | 54 |
| Denver | 50 |

Min DOB: 3/27/1970
Max DOB: 11/8/1995

Customer Details

Jeniffer Allen
Jeniffer236@email.com
10/13/1980 12:00:00 AM

Contact Info:
Home: (123) 556-99999
211 Washington ST
Fort Collins, CO 80642



Name m 2

Brand 3

Items

| Name | Brand |
|---|---|
| Item 20 | Brand 23 |
| Item 24 | Brand 33 |
| Item 26 | Brand 31 |
| Item 28 | Brand 43 |

Running Out

| Store | Item | In Stock |
|---|---|---|
| 1 | 52 | 3 |
| 1 | 31 | 4 |
| 1 | 51 | 4 |
| 2 | 75 | 4 |
| 3 | 1 | 3 |
| 3 | 67 | 3 |
| 3 | 95 | 3 |
| 3 | 5 | 4 |
| 4 | 56 | 3 |
| 4 | 8 | 4 |
| 4 | 19 | 4 |
| 4 | 46 | 4 |
| 5 | 70 | 3 |
| 5 | 79 | 4 |
| 5 | 89 | 4 |
| 6 | 80 | 3 |
| 6 | 35 | 4 |
| 7 | 39 | 4 |
| 7 | 95 | 4 |
| 8 | 7 | 3 |
| 8 | 47 | 3 |
| 8 | 75 | 4 |
| 9 | 33 | 3 |
| 9 | 52 | 4 |
| 10 | 10 | 3 |
| 10 | 26 | 3 |
| 10 | 70 | 3 |
| 10 | 34 | 4 |
| 11 | 3 | 4 |
| 11 | 98 | 4 |
| 12 | 51 | 3 |

Sold

| Store | Item | Sold |
|---|---|---|
| 1 | 20 | 13 |
| 1 | 11 | 8 |
| 1 | 31 | 8 |
| 1 | 73 | 8 |
| 1 | 98 | 7 |
| 1 | 6 | 6 |
| 1 | 66 | 5 |
| 1 | 5 | 4 |
| 1 | 7 | 4 |
| 1 | 21 | 4 |
| 1 | 23 | 4 |
| 1 | 25 | 4 |
| 1 | 34 | 4 |
| 1 | 37 | 4 |
| 1 | 43 | 4 |
| 1 | 48 | 4 |
| 1 | 49 | 4 |
| 1 | 60 | 4 |
| 1 | 68 | 4 |
| 1 | 77 | 4 |
| 1 | 79 | 4 |
| 1 | 80 | 4 |
| 1 | 86 | 4 |
| 1 | 93 | 4 |
| 1 | 95 | 4 |
| 1 | 97 | 4 |
| 1 | 2 | 3 |
| 1 | 14 | 3 |
| 1 | 18 | 3 |
| 1 | 28 | 3 |
| 1 | 39 | 3 |

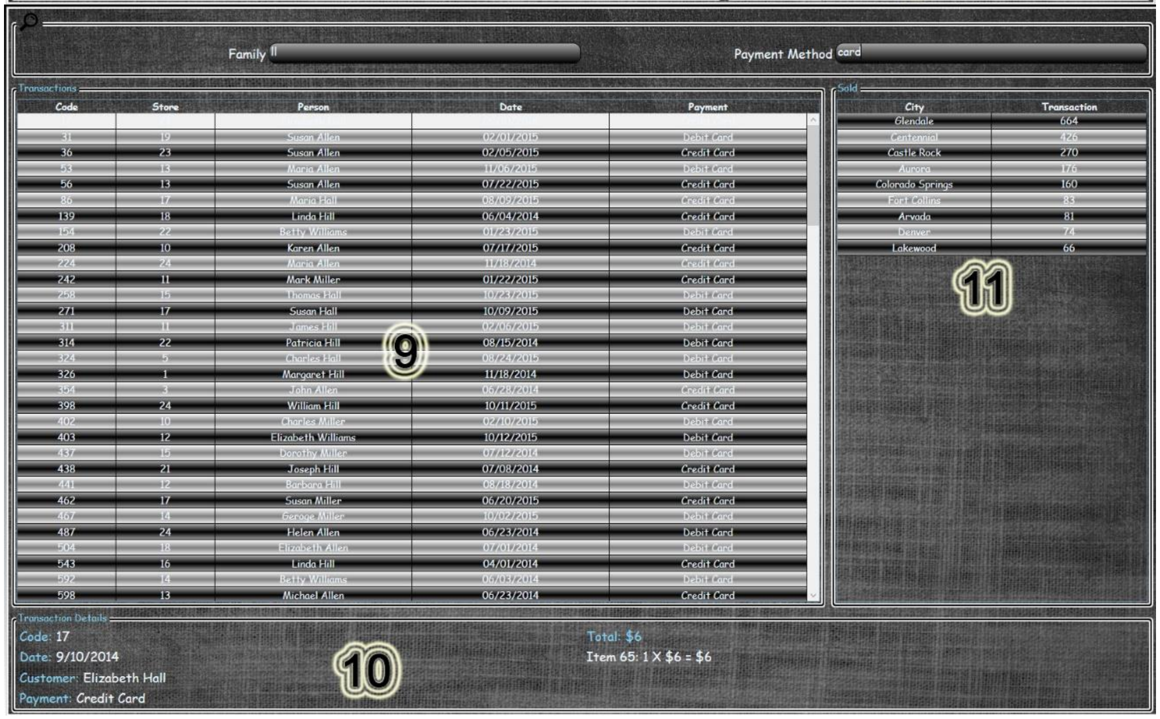
Item Details

Item 26
Brand 31
$84.00

Details:
Material:Corrugated Card+Satin Ribbon
Size:1.9 x3.1 x0.9
12 random colors in one pack

Best/Wors Sellers

Best-Seller Item:
Item 50, Sold: 99
Most Popular Brand:
Brand 13, Sold: 358



Family ll

Payment Method card

Transactions

| Code | Store | Person | Date | Payment |
|---|---|---|---|---|
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 31 | 19 | Susan Allen | 02/01/2015 | Debit Card |
| 36 | 23 | Susan Allen | 02/05/2015 | Credit Card |
| 53 | 13 | Maria Allen | 11/06/2015 | Debit Card |
| 56 | 13 | Susan Allen | 07/22/2015 | Credit Card |
| 86 | 17 | Maria Hall | 08/09/2015 | Credit Card |
| 139 | 18 | Linda Hill | 06/04/2014 | Credit Card |
| 154 | 22 | Betty Williams | 01/23/2015 | Debit Card |
| 208 | 10 | Karen Allen | 07/17/2015 | Credit Card |
| 224 | 24 | Maria Allen | 11/18/2014 | Credit Card |
| 242 | 11 | Mark Miller | 01/22/2015 | Credit Card |
| 258 | 15 | Thomas Hall | 10/23/2015 | Debit Card |
| 271 | 17 | Susan Hall | 10/09/2015 | Debit Card |
| 311 | 11 | James Hill | 02/06/2015 | Debit Card |
| 314 | 22 | Patricia Hill | 08/15/2014 | Debit Card |
| 324 | 5 | Charles Hall | 08/24/2015 | Debit Card |
| 326 | 1 | Margaret Hill | 11/18/2014 | Debit Card |
| 354 | 3 | John Allen | 06/28/2014 | Credit Card |
| 398 | 24 | William Hill | 10/11/2015 | Credit Card |
| 402 | 10 | Charles Miller | 02/10/2015 | Debit Card |
| 403 | 12 | Elizabeth Williams | 10/12/2015 | Debit Card |
| 437 | 15 | Dorothy Miller | 07/12/2014 | Debit Card |
| 438 | 21 | Joseph Hill | 07/08/2014 | Credit Card |
| 441 | 12 | Barbara Hill | 08/18/2014 | Debit Card |
| 462 | 17 | Susan Miller | 06/20/2015 | Credit Card |
| 467 | 14 | Geroge Miller | 10/02/2015 | Debit Card |
| 487 | 24 | Helen Allen | 06/23/2014 | Debit Card |
| 504 | 18 | Elizabeth Allen | 07/01/2014 | Debit Card |
| 543 | 16 | Linda Hill | 04/01/2014 | Credit Card |
| 592 | 14 | Betty Williams | 06/03/2014 | Debit Card |
| 598 | 13 | Michael Allen | 06/23/2014 | Credit Card |

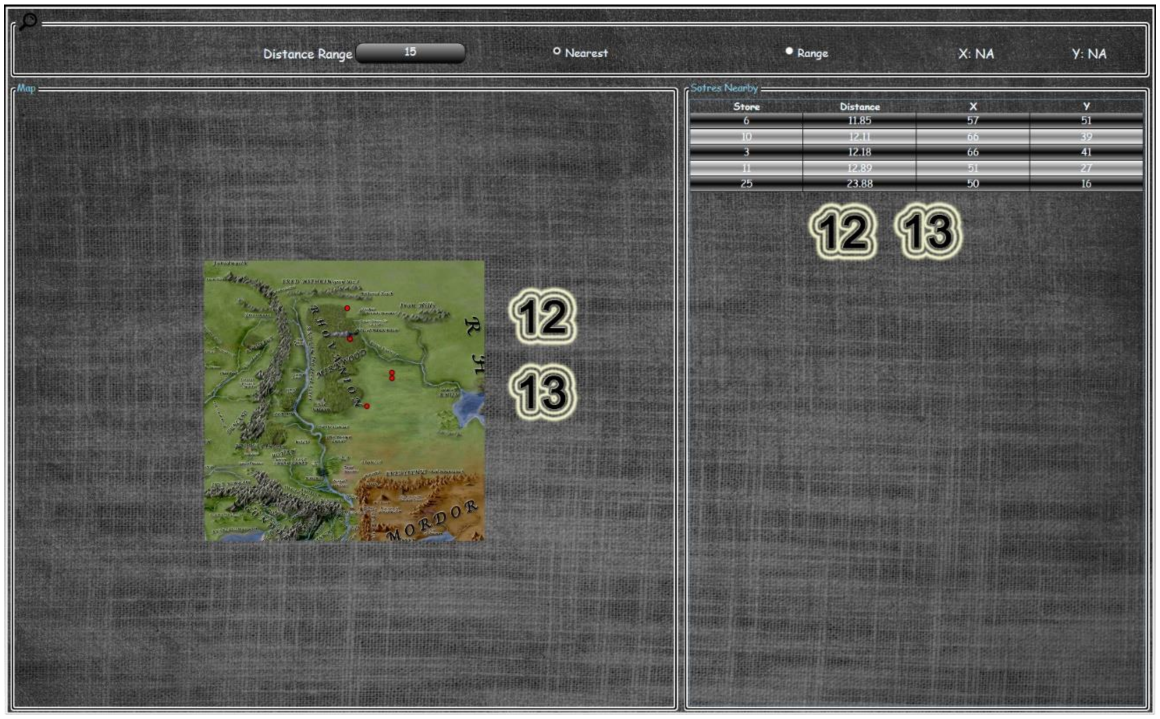
Sold

| City | Transaction |
|---|---|
| Glendale | 664 |
| Centennial | 426 |
| Castle Rock | 270 |
| Aurora | 176 |
| Colorado Springs | 160 |
| Fort Collins | 83 |
| Arvada | 81 |
| Denver | 74 |
| Lakewood | 66 |

Transaction Details

Code: 17
Date: 9/10/2014
Customer: Elizabeth Hall
Payment: Credit Card

Total: $6
Item 65: 1 X $6 = $6



| Store | Distance | X | Y |
|---|---|---|---|
| 6 | 11.85 | 57 | 51 |
| 10 | 12.11 | 66 | 39 |
| 3 | 12.18 | 66 | 41 |
| 11 | 12.89 | 51 | 27 |
| 25 | 23.88 | 50 | 16 |

## Submission Guideline

Please submit your assignment through **Canvas** before the deadline; late submissions are not accepted. You are allowed to submit your assignment multiple times, but only the last submission (**before the deadline**) will be recorded and graded.

Your submission should be a single file named <Your CU Denver Portal ID>_A1-b.zip. For example, if your CU Denver Portal ID is john, the file name would be john_A1-b.zip. The submitted compressed file must contain the following files:

- ***Create***
  This folder contains all CREATE SQL scripts for each table containing the foreign keys and primary key constraints. You have to name them as TABLENAME.sql. For example, the CREATE SQL script of the "PERSON" table should be named as PERSON.sql

- ***DataLayer.cs***
  This file contains the DataLayer class that you have to complete.

- ***SP***
  This folder contains all of your stored procedures (if any). You have to name them as Q_query_number.sql. For example, if you implement a stored procedure for Query #22, it should be named as Q22.sql.

- ***Readme.pdf***
  This file contains your first and last name, CU Denver ID, 9 digit student id, and your UC Denver email address.

**Please download your assignment after submission and check its files to make sure that they are not corrupted.**

**You are highly encouraged to ask your question on Piazza under the "Assignment 1" folder. Please DO NOT include your codes, diagrams and solutions in the comments you share on Piazza. Feel free to help other students with general questions.**