Sp20 CSCI 1550-70 DataBase Management Fundamentals

Table of Contents

[Change Tracking 2](#_Toc38124278)

[Team Members 3](#_Toc38124279)

[Description of Business 4](#_Toc38124280)

[Database Model 4](#_Toc38124281)

[Conceptual Data Modeling 5](#_Toc38124282)

[Logical Data Modeling 8](#_Toc38124283)

[Physical Data Modeling 8](#_Toc38124284)

[Business Roles 9](#_Toc38124285)

[Application Needs 10](#_Toc38124286)

[Employee Data Handling 10](#_Toc38124287)

[Customer Data Handling 10](#_Toc38124288)

[Transaction Handling 10](#_Toc38124289)

[Accounting Reports and other Activity 11](#_Toc38124290)

[Examples of Application Usage 12](#_Toc38124291)

[Database Setup 12](#_Toc38124292)

[Create DDL script 12](#_Toc38124293)

[Alter DDL script 14](#_Toc38124294)

[Load DDL script 15](#_Toc38124295)

[InitAdmin DDL script 16](#_Toc38124296)

[Drop DDL script 17](#_Toc38124297)

[Demo of Application Usage of Database 18](#_Toc38124298)

[Connect to the Database 18](#_Toc38124299)

[User Log in to the Application 20](#_Toc38124300)

[Main Menu screen 22](#_Toc38124301)

[Add Customer 24](#_Toc38124302)

[New Transaction 28](#_Toc38124303)

[Other SQL Queries 32](#_Toc38124304)

[Equi-join SQL Query 32](#_Toc38124305)

[Create a View for SQL Query 33](#_Toc38124306)

[Document Comments 35](#_Toc38124307)

# Change Tracking

|  |  |  |
| --- | --- | --- |
| Date | Changer(s) | Reason(s) |
| 1/31, 2020 | Andy | Initial draft |
| 2/2, 2020 | Andy | Add in some basic data modeling commentary |
| 2/5/2020 | Rosie | Add comments from meeting |
| 02/05/2020 | Mohamed | Add comments how we do the project |
| 02/12/2020 | Rosie | Deleted chair items |
| 02/19/2020 | Andy | Made some document formatting adjustments and a few other comments |
| 2/19/2020 | Rosie | Rosie, Mohamed, & Andy met today.  We made changes as listed in the comments. |
| 4/15/2020 | Andy | Add application details |
|  |  |  |
|  |  |  |

# Team Members

Team members include:

Rosemarie Ruiz

651-361-0179

[Rosemarieruiz25@gmail.com](mailto:Rosemarieruiz25@gmail.com)

[Rosemarie.ruiz@saintpaul.edu](mailto:Rosemarie.ruiz@my.saintpaul.edu)

Mohamed A Ismail

651-447-0464

[Mismail.mo71@gmail.com](mailto:Mismail.mo71@gmail.com)

[Mohamed.ismail.2@my.metrostate.edu](mailto:Mohamed.ismail.2@my.metrostate.edu)

Andrew Roe

651-260-9628

[Cx5314mw@go.minnstate.com](mailto:Cx5314mw@go.minnstate.com)

# Description of Business

The business that we intend to do data modeling for will be retail. The name of the fake business we’ve chosen is “Rosie’s Salon”. It is a Salon that provides hair products and services such as haircuts, shampoos, deep conditioners, and hair-dying.

# Database Model

We will need to discuss this aspect of the project and probably a fair amount time will be spent on this task. Perhaps it will be possible to iteratively cycle through the process of data modeling repeatedly until we are happy with the end result.

There seems to be 3 main phases for data modeling:

1. Conceptual
2. Logical
3. Physical

## Conceptual Data Modeling

Conceptual data modeling is the first step in data modeling and introduces the use of the Entity-Relationship diagram, E-R, or perhaps the Enhanced E-R, EER diagram.

Before we can create a conceptual data model it is necessary to jot down what are the key points of our simulated business. To keep the size of our task achievable in a one-semester class the following assumptions are made:

* There will only be one “Rosie’s Salon” store. It will not be a chain or in any way a business whereby employees might work at multiple locations.
* The accounting needed for the business will be farmed out, not done in house. The only thing needed in our data modeling is a way to generate reports which will be sent to the accounting business contracted to make payroll checks or deposits, withhold payroll taxes and send those taxes to the IRS and state department or revenue and do the needed yearly tax forms.
* Web application costs and any internet access and phone access is all contracted out and not part of the business data model other than that it is a business expense; in much the same way as the contracted-out accounting or other utility expenses. They are treated pretty much like a business expense such as the building lease.
* The employees will include part-time and full-time workers. The only salaried worker(s) is the manager(s). It is anticipated that all employees will be hourly wage earners. This translates into 3 worker job roles; manager, receptionist, and stylist.
* The business will be housed in one location and have 8 chairs.
* The receptionist role can be done by a stylist or a manager. This would be most likely during non-peak business hours or if there are one a few stylist chairs in the salon. When and if the salon gets many chairs or is especially busy then most likely then the receptionist will only be actively a receptionist.
* Employees can take time off – not be available on any given day. The policy for the amount of time away (time off or unavailable to be scheduled) will be dictated by written policy. But the data model will only need to concern itself with tracking what time away has been taken or is scheduled in the future.
* Credit charge tips are tracked by the business and will be in the business data model.
* Customers can just stop in the buy products or walk in for service or check in over an internet web App.

Given those assumptions the entities and the relationship of those entities is as follows:

1. Employee (EmpID, CreatedTimeDate, LastUpdatedTimeDate, FirstName, MiddleInitial, LastName,Phone,Email)
2. EmployeeHistory (EmpHistID, EmpID, UpdaterEmpID, ActivateIndicator, LastUpdatedTimeDate, FirstName, MiddleInitial, LastName,Phone,Email)
3. Customer (CustID, CreatedTimeDate, LastUpdatedTimeDate, FirstName, MiddleInitial, LastName,Phone,Email,AlternateContact)
4. CustomerHistory (CustHistID, CustID, UpdaterEmpID, ActivateIndicator, LastUpdatedTimeDate, FirstName, MiddleInitial, LastName,Phone,Email)
5. Product (ProductID, CreatedTimeDate, LastUpdatedTimeDate, NumberInStock, PhysicalLocation, ProductDescription)
6. \*\*ProductHist (ProductID, UpdaterEmpID, LastUpdatedTimeDate, BackOrderedIndicator, numberOnOrder, VendorID)
7. Service(ServiceID, CreatedTimeDate, LastUpdatedTimeDate, ServiceDescription)
8. \*\*Vendor (VendorID, CreatedTimeDate, LastUpdatedTimeDate, Business, BusinessAddress, Phone,Email)
9. \*\*VendorProducts (VendorID, ProductID, UpdaterEmpID, LastUpdatedTimeDate)
10. ServiceProvider (ServiceProviderID, CreatedTimeDate, LastUpdatedTimeDate, Business, BusinessAddress, Phone, Email)
11. ServiceProviderHist (ServiceProviderHistID, ServiceProviderID, UpdaterEmpID, LastUpdatedTimeDate, FirstName, MiddleInitial, LastName, Phone, Email, DateTimeLastPaid, AmountLastPaid, OutstandingBalance)

(Things that have \*\* are optional and may be listed in the assignment when our main part is done)

Nore: UpdaterEmpID is equal to an EmpID.

The relationships of these entities is:

1. Employee to EmployeeInfo is one to many,
2. Customer to CustomerHist is one to many,
3. Product to ProductHist is one to many,
4. Vendor to VendorProducts is one to many,
5. Product to ProductHist is one to many,
6. ServiceProvider to ServiceProviderHist is one to many

To generate the E-R diagram for this set of entities and relationships is something I’m struggling with at this time. But will be included here when it becomes available.

## Logical Data Modeling

Logical data modeling is the second major step in data modeling. In this modeling step the Conceptual data model is “normalized”. See page 177 of the 11th edition of the Modern Database Management book. Key concepts include:

1. 1NF – First Normal Form
2. 2NF – Second Normal Form
3. 3NF – Third Normal Form
4. BCNF – Boyce-Codd Normal Form
5. 4NF – Fourth Normal Form
6. 5NF – Fifth Normal Form

Of these 2NF seems to be the most desired “normalization” form.

At any rate the E-R model created during the conceptual phase needs to be translated into a relational model.

The Logical E-R model for this project is in a separate Power Point document. Please refer to that for the DB model details.

## Physical Data Modeling

Physical data modeling is the third and final data modeling step. This step can result in “denormalization”, whereby vertical or horizontal partitioning is used for combining certain entities or possibly re-creating redundant attributes of the entities, which had been removed during the “normalization” logical data modeling phase. This would mainly be done in order to improve data access and updating efficiency.

This last step of data modeling refers to the relational model to help create the physical data model. From this physical model it should be very possible to create the DDL, Data Definition Language, SQL statements needed to create the desired Database Schema and tables.

# Business Roles

There are several business roles:

* Manager
* Stylist
* Receptionist
* Accountant or Financial officer
* Data Engineer

Each of these roles can translate into a different view of the business data. For instance, only the manager(s) and probably the accountant(s) should have access to other employee’s salary or commission information. For this reason, it is likely that various database views may be desirable.

# Application Needs

During the course of business for Rosie’s Salon several Application needs should be satisfied. This includes:

* Employee information handling
* Customer information handling
* Transaction processing
* Accounting reports and activities

## Employee Data Handling

There is a need for an interface tool to add, delete and update employee information. This could include:

* Employee name; create or update
* Employee contact information; phone(s), mailing address, email address(es), W2 info, salary or commission, job role, etc.
* Employee timecard info; each start time and duration

## Customer Data Handling

Also, is a need for an interface tool to add, delete and update customer information. This could include:

* Customer name; create or update
* Customer contact information; phone(s), mailing address, email address(es), perhaps some short (less than 256 characters) of customer comments or feedback, etc.
* Customer service info; each start time and duration, what service was provided, e.g. hair syle, manicure, etc.

## Transaction Handling

This is a biggie – a lot goes into processing a transaction. This could include:

* Customer identifier
* Time stamp of transaction
* Products sold to customer
* Services rendered to customer
* Cost of each product or service provided
* Total cost to customer of this transaction
* Employee identified for any and all services provided to customer
* Payment method customer used
* In a later phase of project, maybe various actions that were precipitated due to this transaction. For example, maybe some product is almost out and so an order needs to be placed for more of that product.
* Many other pieces of information are possible.

## Accounting Reports and other Activity

There is a need for an interface tool to help handle accounting types of activities. This could include:

* Producing a paycheck
* Withholding handling for Federal, State and local taxes
* Withholding for Social Security, Unemployment
* Automatic contribution handling for any retirement plan (401k, or whatever)
* Final year end reports for the business to analyze
* Tax year reports needed by the various tax entities

# Examples of Application Usage

In order to better demonstrate the design of the Rosie’s Salon database it is helpful to dive into details of how that can be done with actual examples. This, is it hoped, will demonstrate the viability of the database design model? By illustrating a portion of the setup, employee handling, customer handling and transaction handling the reader should be able to gain confidence that indeed this data model is viable for business being modeled.

ng the course of business for Rosie’s Salon several Application needs should be satisfied. This includes:

* Employee information handling
* Customer information handling
* Transaction processing
* Accounting reports and activities

## Database Setup

Before any of the employee, customer, transaction or report processing can be attempted a database has to be set up.

There were 2 different database engines used in testing and demo-ing this DB design - Oracle via Apex and MySQL. Scripts were used for the Data Definition Language (DDL) SQL commands to do the setup. The scripts for Oracle and MySQL DDL were very similar but with slight differences. This document will discuss each DDL setup script in general and then also point out the unique differences needed for handling each type of database engine used.

### Create DDL script

The script to create the tables was:

CREATE TABLE Employee (

EmpID NUMBER NOT NULL,

UserID NUMBER,

CreateTime VARCHAR2(32),

UpdateTime VARCHAR2(32),

Fname VARCHAR2(64),

Minit VARCHAR2(64),

Lname VARCHAR2(64),

job VARCHAR2(16),

salary NUMBER(15,2),

commission NUMBER(15,2)

);

CREATE TABLE Customer (

CustID NUMBER NOT NULL,

UserID NUMBER,

CreateTime VARCHAR2(32),

UpdateTime VARCHAR2(32),

Fname VARCHAR2(64),

Minit VARCHAR2(64),

Lname VARCHAR2(64),

phone VARCHAR2(16),

email VARCHAR2(64)

);

Etc.

CREATE TABLE EmpInfo (

EinfoID NUMBER NOT NULL,

EmpID NUMBER,

UserID NUMBER,

UpdateTime VARCHAR(32),

InfoType NUMBER,

InfoSubType NUMBER,

Validity NUMBER,

Nbr1Parm NUMBER,

Nbr2Parm NUMBER(15,2),

CharBig VARCHAR(64)

);

CREATE TABLE CustInfo (

CinfoID NUMBER NOT NULL,

CustID NUMBER,

UserID NUMBER,

UpdateTime VARCHAR(32),

InfoType NUMBER,

InfoSubType NUMBER,

Validity NUMBER,

Nbr1Parm NUMBER,

Nbr2Parm NUMBER(15,2),

CharBig VARCHAR(64)

);

Etc.

For the Oracle database.

For the MySQL database, the same functionality was:

USE rosiessalon;

CREATE TABLE Employee (

EmpID INT NOT NULL AUTO\_INCREMENT,

UserID INT,

CreateTime VARCHAR(32),

UpdateTime VARCHAR(32),

Fname VARCHAR(64),

Minit VARCHAR(64),

Lname VARCHAR(64),

job VARCHAR(16),

salary DECIMAL(15,2),

commission DECIMAL(15,2),

PRIMARY KEY (EmpID)

);

Etc.

CREATE TABLE EmpInfo (

EinfoID INT NOT NULL AUTO\_INCREMENT,

EmpID INT,

UserID INT,

UpdateTime VARCHAR(32),

InfoType INT,

InfoSubType INT,

Validity INT,

Nbr1Parm INT,

Nbr2Parm DECIMAL(15,2),

CharBig VARCHAR(64),

PRIMARY KEY (EinfoID)

);

Etc.

Note that the MySQL database script, needed to select the database being used with a “USE <dbname>” statement.

Note also that the Oracle version used “NUMBER” where the MySQL used “INT”.

### Alter DDL script

The script to alter the tables was:

For the Oracle DB -

ALTER TABLE Employee ADD CONSTRAINT Cemployee\_PK PRIMARY KEY ( EmpID );

ALTER TABLE Employee ADD CONSTRAINT Cemployee\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

ALTER TABLE Customer ADD CONSTRAINT Ccustomer\_PK PRIMARY KEY ( CustID );

ALTER TABLE Customer ADD CONSTRAINT Ccustomer\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

Etc.

And then for the auto-increment -

CREATE SEQUENCE EmpID\_Seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 5;

CREATE SEQUENCE CustID\_Seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 5;

CREATE SEQUENCE ProdID\_Seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 5;

CREATE SEQUENCE ServID\_Seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 5;

CREATE SEQUENCE SuppID\_Seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 5;

And for MySQL DB -

ALTER TABLE Employee ADD CONSTRAINT Cemployee\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

ALTER TABLE Product ADD CONSTRAINT Cproduct\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

ALTER TABLE Service ADD CONSTRAINT Cservice\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

ALTER TABLE Supplier ADD CONSTRAINT Csupplier\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

ALTER TABLE Transaction ADD CONSTRAINT Ctrans\_EmpID\_FK FOREIGN KEY (CustID) REFERENCES Customer (CustID);

ALTER TABLE Transaction ADD CONSTRAINT Ctrans\_User\_FK FOREIGN KEY (UserID) REFERENCES Employee (EmpID);

Etc.

The CREATE SEQUENCE statement was not needed for MySQL.

### Load DDL script

The script to alter the tables was:

For the Oracle DB -

INSERT INTO Employee (EmpID, UserID, CreateTime, UpdateTime, Fname, Minit, Lname, job, salary, commission)

VALUES (EmpID\_Seq.nextval, 1, '20200102000001', '20200102000001', 'Rosemarie', 'A', 'Ruiz',

'Manager', 20.00, 0.00);

INSERT INTO Employee (EmpID, UserID, CreateTime, UpdateTime, Fname, Minit, Lname, job, salary, commission)

VALUES (EmpID\_Seq.nextval, 1, '20200102000001', '20200102000001', 'Mohamed', 'A', 'Ismail',

'Stylist', 10.00, 0.00);

INSERT INTO Employee (EmpID, UserID, CreateTime, UpdateTime, Fname, Minit, Lname, job, salary, commission)

VALUES (EmpID\_Seq.nextval, 1, '20200102000001', '20200102000001', 'Andrew', 'L', 'Roe',

'DataEngineer', 200.00, 1000000.00);

INSERT INTO Employee (EmpID, UserID, CreateTime, UpdateTime, Fname, Minit, Lname, job, salary, commission)

VALUES (EmpID\_Seq.nextval, 1, '20200102000001', '20200102000001', 'Khiet', 'A', 'Nguyan',

'Receptionist', 10.00, 1000.00);

INSERT INTO EmpInfo (EinfoID, EmpID, UserID, UpdateTime, InfoType, InfoSubType, Validity, CharBig)

VALUES (EinfoID\_Seq.nextval,

(SELECT EmpID FROM Employee WHERE FNAME='Rosemarie' AND MINIT='A' AND LNAME='Ruiz'),

1, '20200102000001', 0, 0, 4, 'rosie');

INSERT INTO EmpInfo (EinfoID, EmpID, UserID, UpdateTime, InfoType, InfoSubType, Validity, CharBig)

VALUES (EinfoID\_Seq.nextval,

(SELECT EmpID FROM Employee WHERE FNAME='Rosemarie' AND MINIT='A' AND LNAME='Ruiz'),

1, '20200102000001', 0, 1, 4, 'rosiepwd');

Etc.

And for MySQL DB -

It was the same. Keep in mind with MySQL, each script does need the

USE ROSIESSALON:

Statement at the beginning (or whatever DB name used can be substituted for ROSIESSALON).

### InitAdmin DDL script

This script is a hybrid of the load DDL script and was used to only load an Admin user. This was so that a GUI App to “load” employees and customers and create transactions. This App requires at one user be able to login first and thus the need to “load” an Admin user into the Employee table of the database.

For either DB engine -

INSERT INTO Employee (UserID, CreateTime, UpdateTime, Fname, Minit, Lname, job, salary, commission)

VALUES (1, '20200101000001', '20200101000001', 'Admin', 'A', 'Admin',

'Administrator', 200.00, 1000000.00);

INSERT INTO EmpInfo (EmpID, UserID, UpdateTime, InfoType, InfoSubType, Validity, CharBig)

VALUES (1, 1, '20200101000000', 0, 0, 4, 'admin');

INSERT INTO EmpInfo (EmpID, UserID, UpdateTime, InfoType, InfoSubType, Validity, CharBig)

VALUES (1, 1, '20200101000000', 0, 1, 4, 'password');

### Drop DDL script

Finally, a script to drop all of the tables can be handy:

For the Oracle DB engine -

DROP TABLE EMPINFO;

DROP TABLE CUSTINFO;

DROP TABLE SUPPLIERINFO;

DROP TABLE PRODUCTINFO;

DROP TABLE TRANSACTIONDETAILS;

DROP TABLE SUPPLIER;

DROP TABLE PRODUCT;

DROP TABLE SERVICE;

DROP TABLE TRANSACTION;

DROP TABLE CUSTOMER;

DROP TABLE EMPLOYEE;

DROP SEQUENCE EmpID\_Seq;

DROP SEQUENCE CustID\_Seq;

DROP SEQUENCE ProdID\_Seq;

DROP SEQUENCE ServID\_Seq;

DROP SEQUENCE SuppID\_Seq;

DROP SEQUENCE TransID\_Seq;

DROP SEQUENCE EinfoID\_Seq;

DROP SEQUENCE CinfoID\_Seq;

DROP SEQUENCE SinfoID\_Seq;

DROP SEQUENCE PinfoID\_Seq;

DROP SEQUENCE TinfoID\_Seq;

For the MySQL engine -

Really the same script minus the “DROP SEQUENCE” statements.

## Demo of Application Usage of Database

A portion of an Application was created using the MySQL DB to demonstrate application usage of this database design. Portions implemented included employee, customer and transaction processing. Report processing and loading or adding products or services was not implemented. There just wasn’t enough time. Plus, the portion completed seems adequate to demonstrate viability of the DB design.

The application was written in Java.

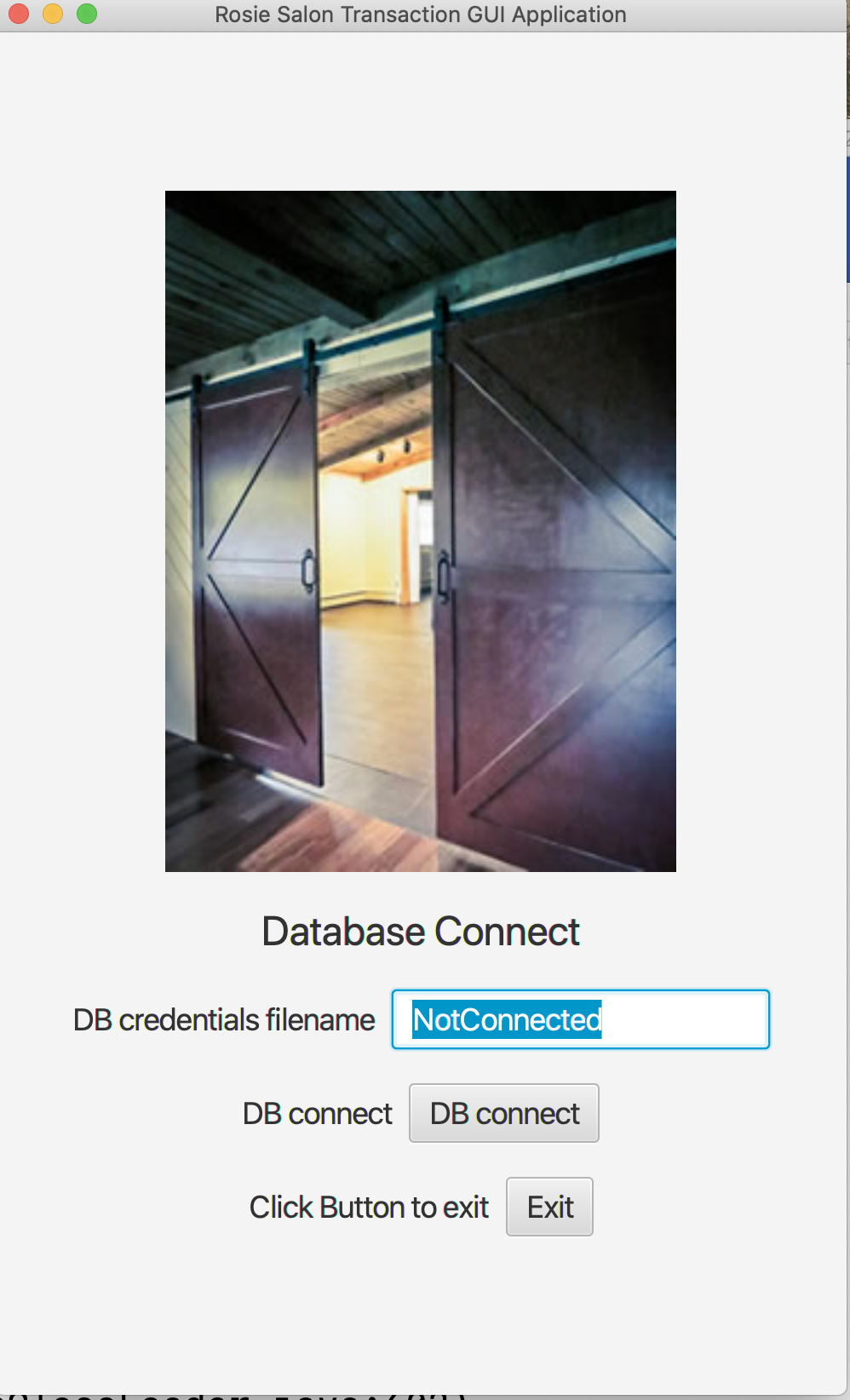
### Connect to the Database

To use the MySQL database, the user needed to make a connection to the database Java DB Connection (JDBC) handling. In order to hide at least some of the credentials used for this the App was designed to receive a small text file with that into for DB connect processing. The format of that file script on the Mac host where the App was tested was:

jdbc:mysql://localhost:3306/salondb,username,userpwd

In this example the JDBC is directed to use port 3306 for connecting with MySQL. The database name was “salondb”. The SQL user is shown as “username” with a SQL password of “userpwd”. A user needs to match the credentials to their actual installation and setup.

Below is how connecting to the DB via the App appears to the user. It is a good lead in to getting a feel for the App GUI interface, which is often pretty simplistic but again was hurriedly put together to try to demonstrate that this database design is viable.



The user is prompted to enter a file name that contains the MySQL connection credentials. When they do so and it has the correct credentials and the user selects “DB connect”, the App then prompts for the user to log in.

### User Log in to the Application

After connecting to the database engine, a user can log into the App. The user must be in the employee table and have username and password records in the EmpInfo table, e.g.

mysql> select EinfoID,EmpID,InfoType,InfoSubType,CharBig from EmpInfo where Validity <> 0 and EmpID = 1;

+---------+-------+----------+-------------+----------+

| EinfoID | EmpID | InfoType | InfoSubType | CharBig |

+---------+-------+----------+-------------+----------+

| 1 | 1 | 0 | 0 | admin |

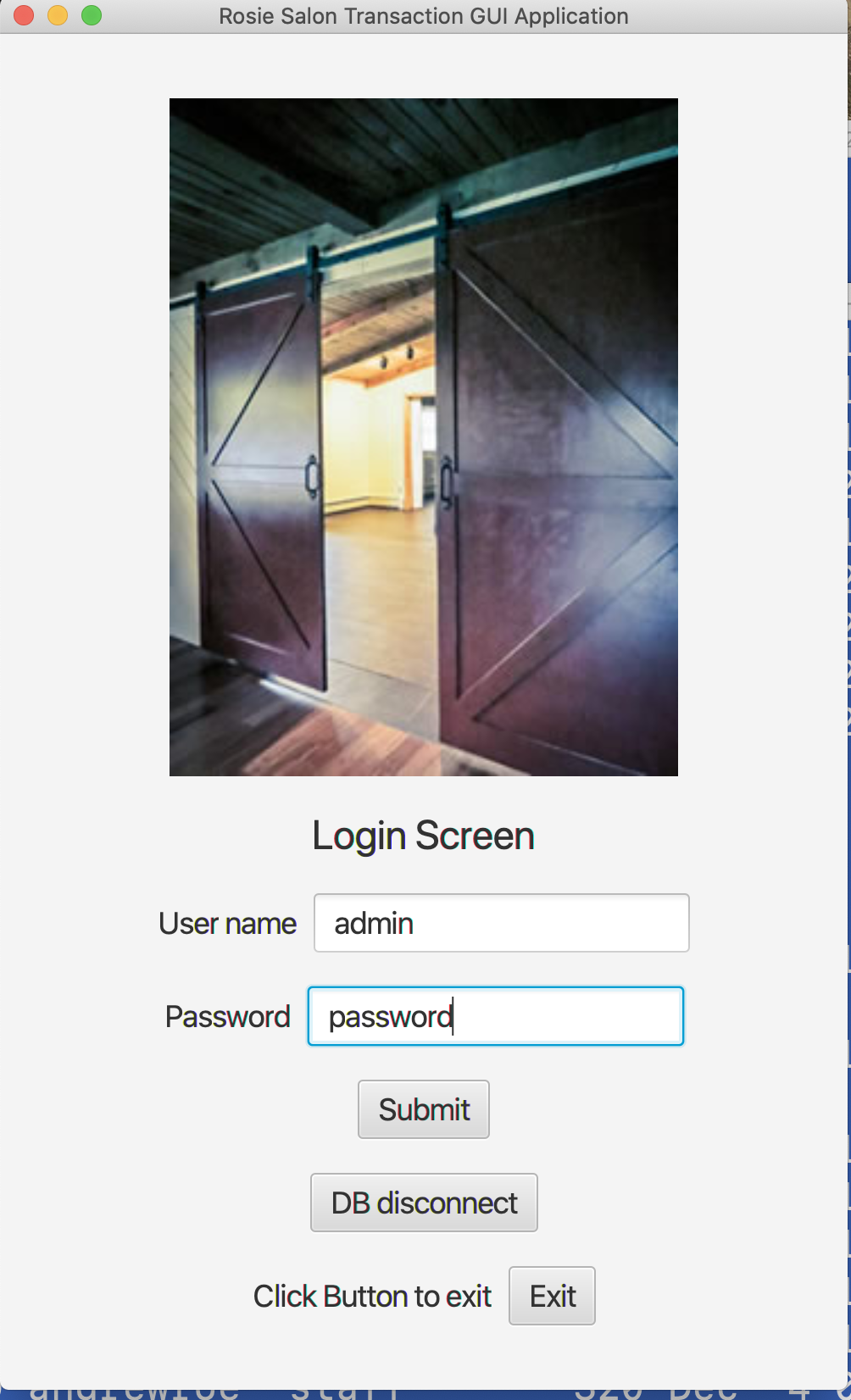
| 2 | 1 | 0 | 1 | password |

+---------+-------+----------+-------------+----------+

2 rows in set (0.01 sec)

Which shows that Employee with ID (PK) = 1, has a username of “admin” and a password or “password”. (Note: at this time the App has no encryption handling for the password. That is left for a later version).

Below we see the Login screen.



The Username of ‘admin’ and Password of ‘password’ have been filled in by the user in this image. After the user selects “Submit”, that “admin” user will be logged in as the App user. That app user is a key concept in the DB design. The reader will notice “UserID” in many tables. It is an employee ID of what user was using the App. Thus, when more employees or customers are added or modified, that “UserID” can also be captured in the DB record for the employee or customer. Similarily when transactions are created by the user logged into the App, the corresponding “UserID” for that logged employee will be tracked in the Transaction and/or TransactionDetails tables.

The SQL statements used to check the login of a user are shown below:

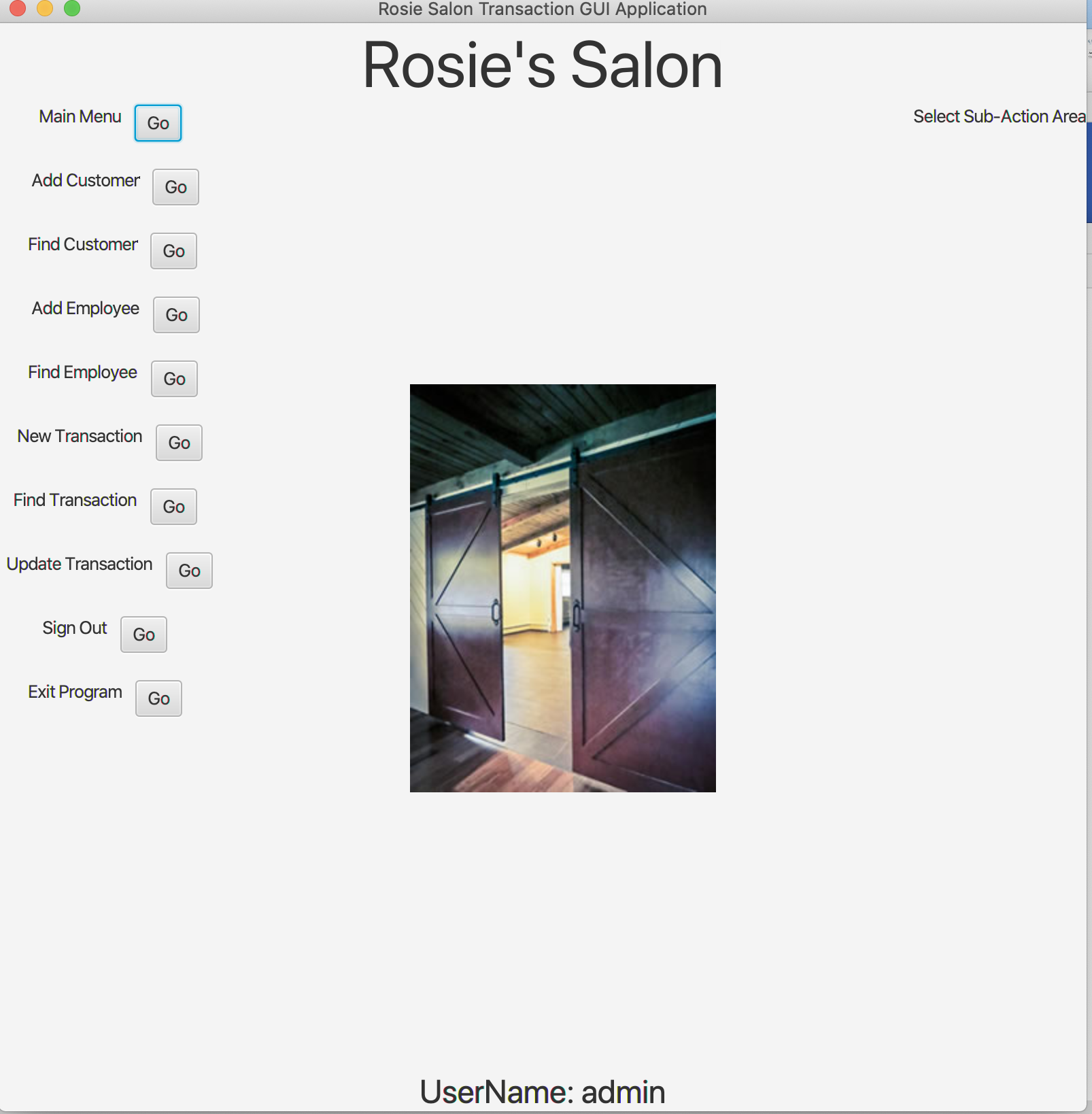
SELECT EmpID, Validity FROM EmpInfo WHERE InfoType = 0 AND InfoSubType = 0 AND Validity <> 0 AND CharBig = 'admin'

in EmpDBaccess - fetchPassword

SELECT Validity, CharBig FROM EmpInfo WHERE EmpID = 1 AND InfoType = 0 AND InfoSubType = 1 AND Validity <> 0

### Main Menu screen

After connecting to the database and logging in as an application user. The App user is presented with a rather crude menu screen (please forgive this simplistic screen. It is on the “to do” list.) See below:

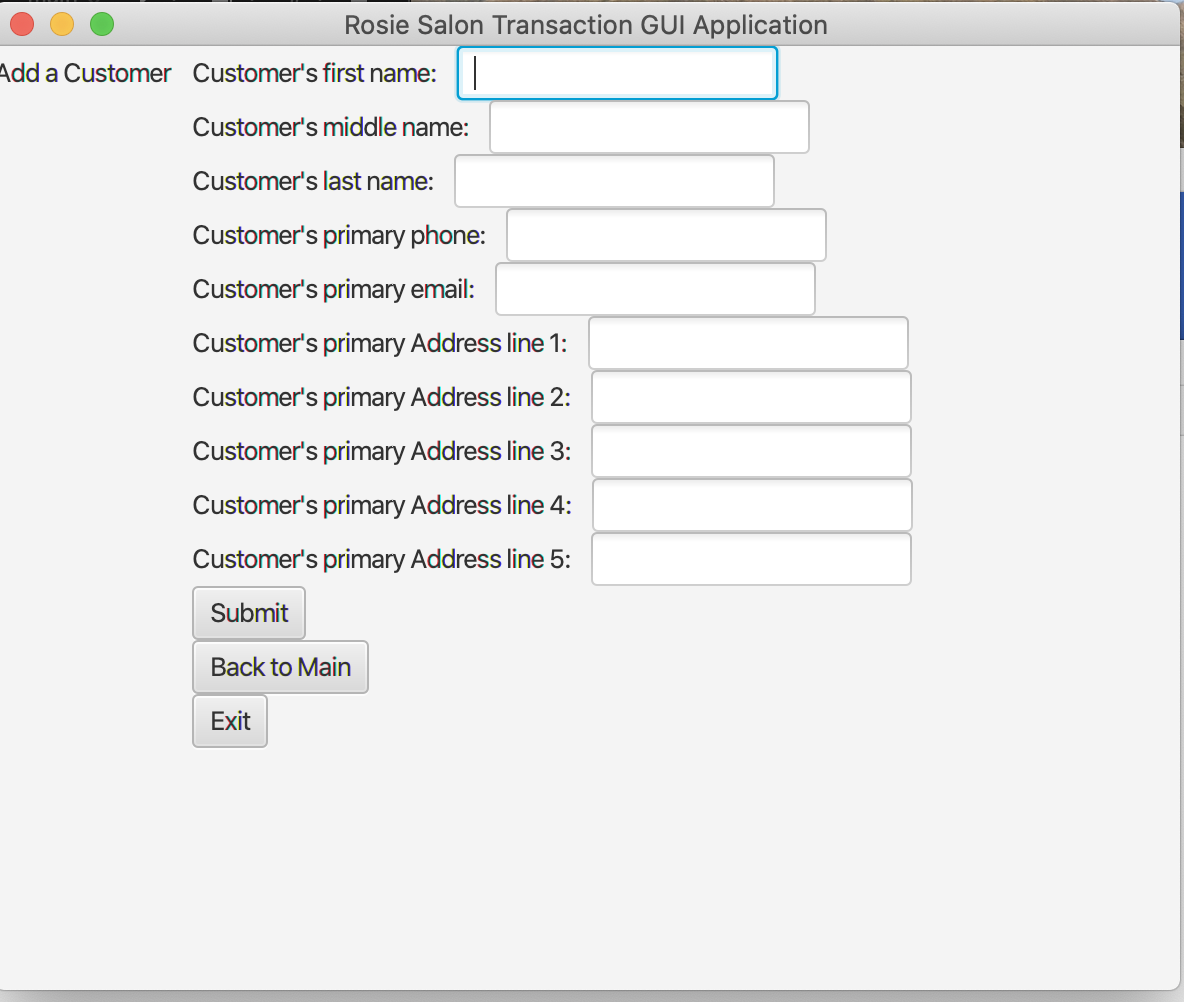


Again, please forgive some of the display's faults, (it just hasn’t been cleaned up yet. One can see the “Select Sub-Action Area” in the upper right. That needs to be removed. It has nothing to do with demoing this App). You can see that the UserName is admin. It would be different if a different user was logged in.

To complete the demo of this DB access and usage, adding a customer and then executing a transaction will be shown in the next sub-sections.

### Add Customer

This is the screen after selecting “Add Customer”.



In this example, Tammy K Jones was filled in for the customer, the primary phone was 612-111-4567 and the primary email was [pk.jones@mymail.net](mailto:pk.jones@mymail.net)

This is fictitious of course. The SQL DML statement to insert this new customer into the DB was:

INSERT INTO Customer (UserID, CreateTime, UpdateTime, Fname, Minit, Lname, phone, email) VALUES (1,'20200418123438', '20200418123438', 'Tammy', 'K', 'Jones', '612-111-4567', '[tk.jones@mymail.net](mailto:tk.jones@mymail.net)')

A new entry in the Customer table has been added for Tammy -

mysql> select \* from Customer;

+--------+--------+----------------+----------------+-------+-------+----------+--------------+----------------------+

| CustID | UserID | CreateTime | UpdateTime | Fname | Minit | Lname | phone | email |

+--------+--------+----------------+----------------+-------+-------+----------+--------------+----------------------+

| 1 | 1 | 20200406194653 | 20200409191902 | Jenn | R | Snow | 763-294-1300 | j.snow2gmail.com |

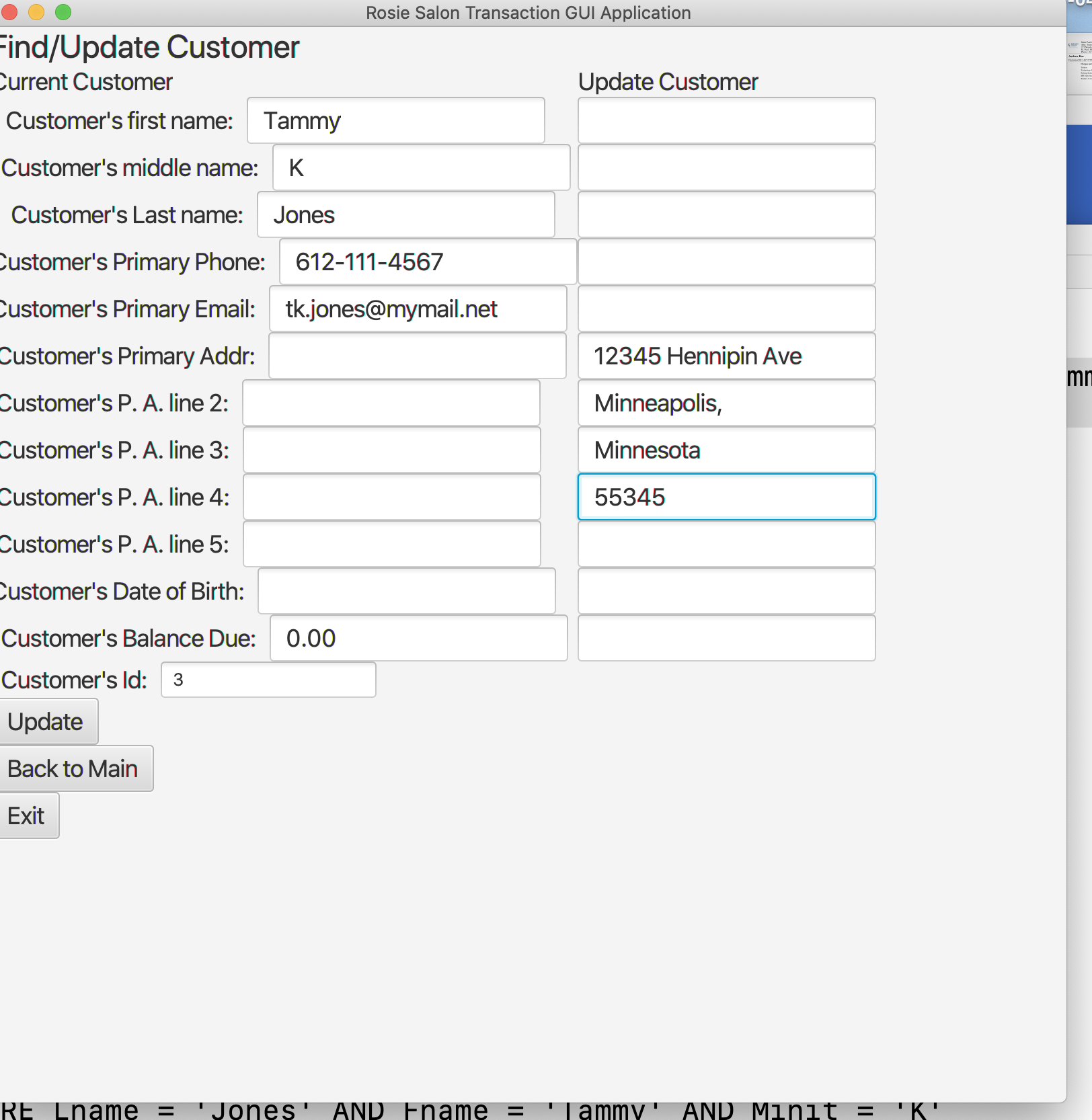
| 2 | 2 | 20200409191652 | 20200409191652 | Pam | A | Williams | 612-324-9786 | [p.williams@otter.net](mailto:p.williams@otter.net) |

| 3 | 1 | 20200418123438 | 20200418123438 | Tammy | K | Jones | 612-111-4567 | [tk.jones@mymail.net](mailto:tk.jones@mymail.net) |

+--------+--------+----------------+----------------+-------+-------+----------+--------------+----------------------+

3 rows in set (0.00 sec)

We can return to the main menu and add some more info for this customer; say an address.



The Find Customer was selected from the Main Menu and some address fields have been filled in for Tammy. Then when Update is selected -

INSERT INTO CustInfo (CustID, UserID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, CharBig) VALUES (3, 1, '20200418124447', 3, 1, 5, 0, '12345 Hennipin Ave')

INSERT INTO CustInfo (CustID, UserID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, CharBig) VALUES (3, 1, '20200418124447', 3, 1, 5, 1, 'Minneapolis, ')

INSERT INTO CustInfo (CustID, UserID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, CharBig) VALUES (3, 1, '20200418124447', 3, 1, 5, 2, 'Minnesota')

INSERT INTO CustInfo (CustID, UserID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, CharBig) VALUES (3, 1, '20200418124447', 3, 1, 5, 3, '55345')

SELECT CustID, UserID, Fname, Lname, Minit, phone, email FROM Customer where CustID = 3

SELECT CustID, UserID, InfoType, InfoSubType, Validity, Nbr1Parm, Nbr2Parm, CharBig FROM CustInfo where CustID = 3

A lot of activity with the database takes place. We see 4 INSERT DML statements to put 4 address lines into the CustInfo table. Then we see 2 SELECT DML statements, which were used by the App to be able to re-display the new information about that customer.

We see new entries added into the CustInfo table for Customer, who’s ID is 3 – Tammy's. Manually running a DML SQL command shows:

mysql> select CinfoID,InfoType,InfoSubType,Validity,Nbr1Parm,CharBig from CustInfo where CustID = 3;

+---------+----------+-------------+----------+----------+--------------------+

| CinfoID | InfoType | InfoSubType | Validity | Nbr1Parm | CharBig |

+---------+----------+-------------+----------+----------+--------------------+

| 11 | 3 | 1 | 5 | 0 | 12345 Hennipin Ave |

| 12 | 3 | 1 | 5 | 1 | Minneapolis, |

| 13 | 3 | 1 | 5 | 2 | Minnesota |

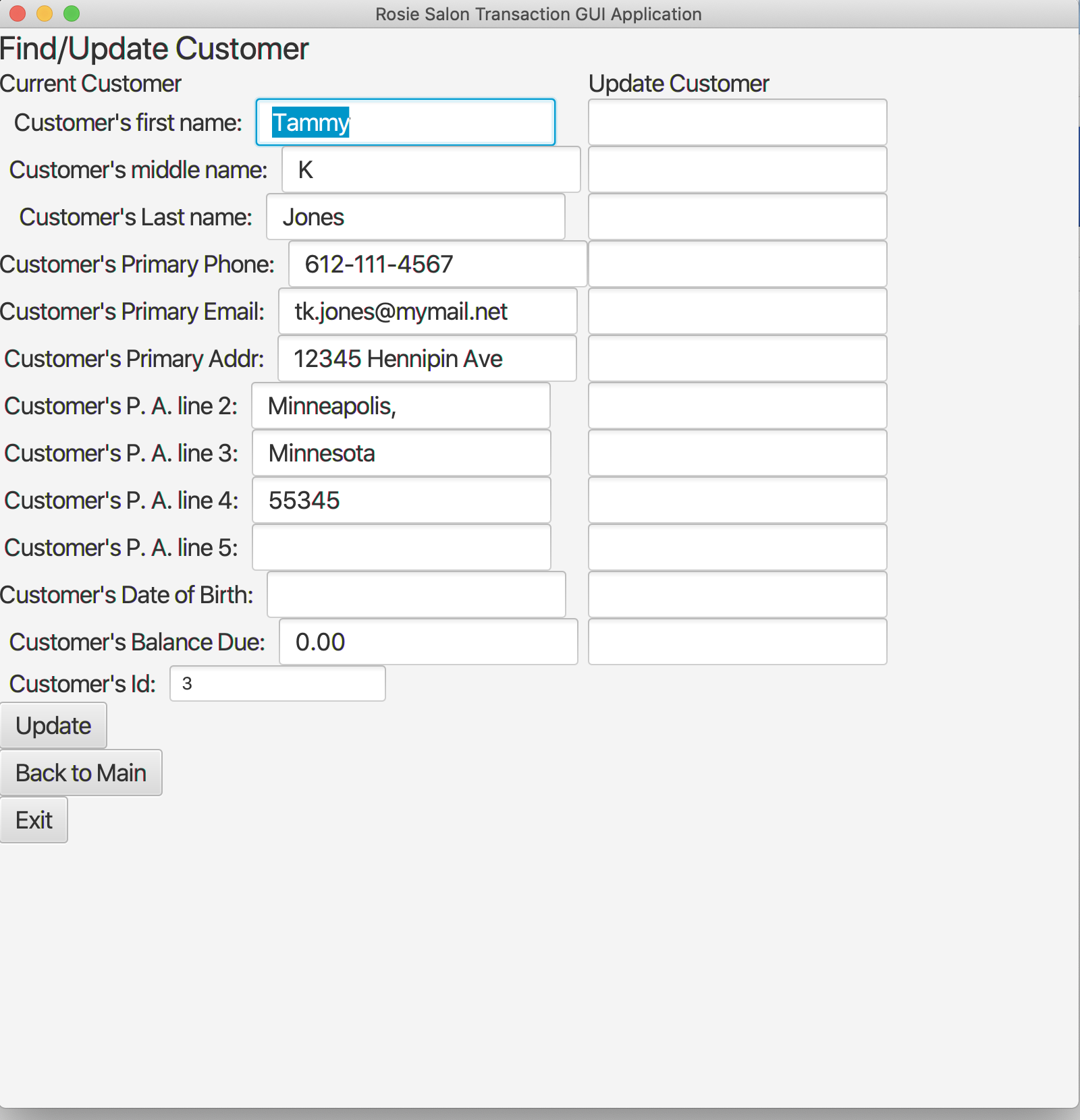
| 14 | 3 | 1 | 5 | 3 | 55345 |

+---------+----------+-------------+----------+----------+--------------------+

4 rows in set (0.00 sec)

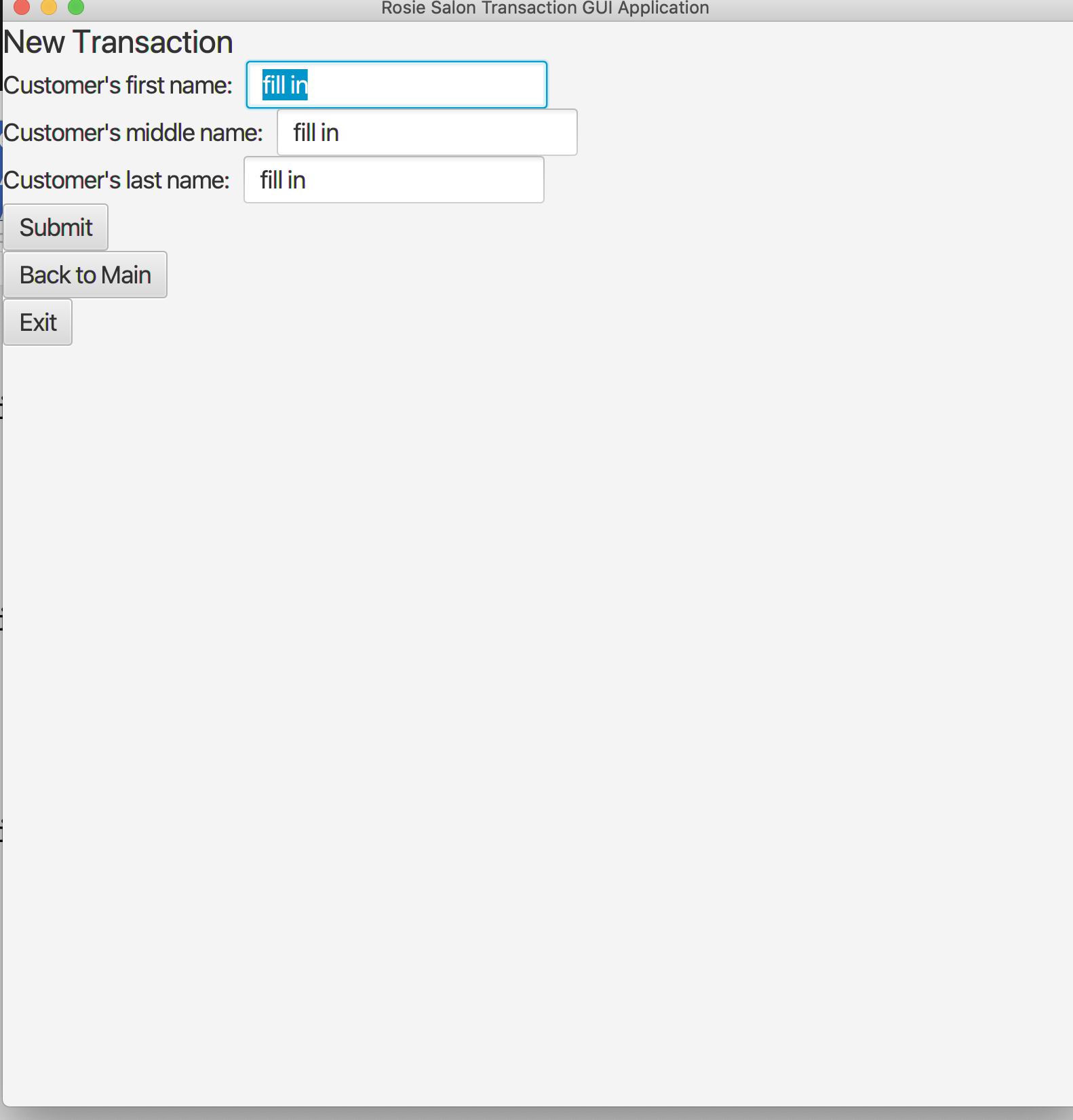
We can see that there are 4 new CinfoID primary key entries in that table for customer of CustID = 3. We see those record entries have an InfoType of 3, an InfoSubType of 1, a Validity of 5, Nbr1Parm of 0, then 1, then 2, and then 3, and finally CharBig values for each of the 4 addresses added. Note that the Validity column value is 5. This is a bit map of 1 + 4, for the first parameter, Nbr1Parm, and the third parameter, CharBig. If the second parameter, Nbr2Parm was valid, a value of 2 would also be included in the Validity field. Another consideration; if the Validity column value was set to 0, then this record is no longer valid – it has been effectively deleted. But for this database design, records are not deleted until end of year processing. Instead deleted information is simply marked invalid by updating the Validity column to become 0.

The user’s App display for Find/Update Customer is now updated to:

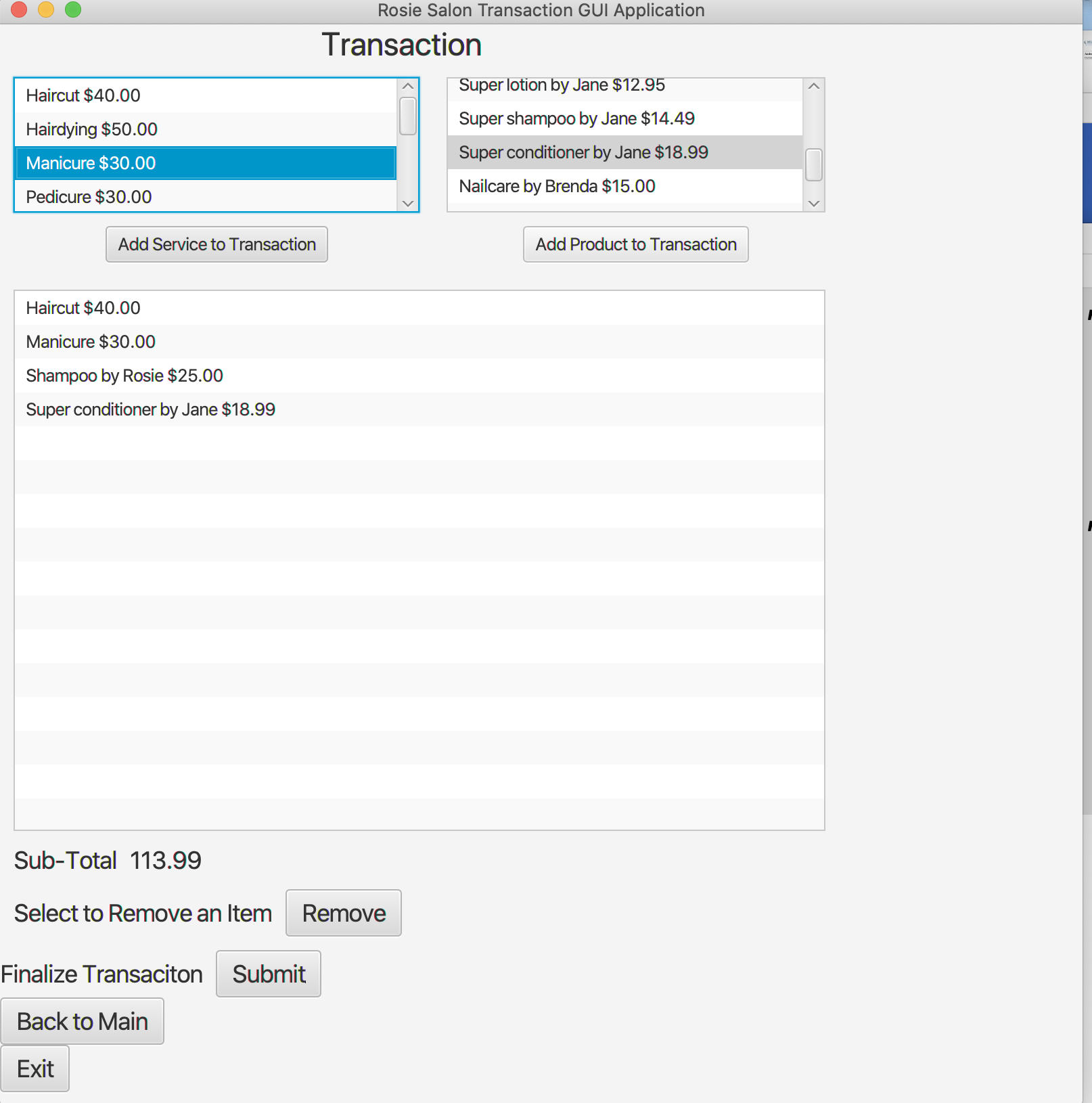


### New Transaction

This is the screen after selecting “New Transaction”.



It requires that the user fil in the Customer’s name and click “Submit”. For this demo, let’s assume she wants a haircut, a manicure, some shampoo and some conditioner.



We can see that the total is $113.99. So, when the user clicks on “Submit” there will be SQL DML statements executed to add the transaction into the database.

From prior testing the Transaction and TransactionDetails have been added to already.

mysql> select \* from Transaction;

+---------+--------+--------+----------------+----------------+--------+------------+--------+

| TransID | CustID | UserID | CreateTime | UpdateTime | Method | CreditInfo | amount |

+---------+--------+--------+----------------+----------------+--------+------------+--------+

| 1 | 1 | 1 | 20200406194744 | 20200406194744 | 1 | NULL | 60.48 |

| 2 | 2 | 2 | 20200409192334 | 20200409192334 | 1 | NULL | 59.49 |

+---------+--------+--------+----------------+----------------+--------+------------+--------+

2 rows in set (0.02 sec)

mysql> select \* from TransactionDetails;

+---------+---------+--------+----------------+----------+-------------+----------+----------+----------+-----------------------------+

| TinfoID | TransID | UserID | UpdateTime | InfoType | InfoSubType | Validity | Nbr1Parm | Nbr2Parm | CharBig |

+---------+---------+--------+----------------+----------+-------------+----------+----------+----------+-----------------------------+

| 1 | 1 | 1 | 20200406194744 | 6 | 1 | 6 | 0 | 29.99 | Mini-Cut |

| 2 | 1 | 1 | 20200406194744 | 6 | 1 | 6 | 0 | 14.50 | Hair Wash |

| 3 | 1 | 1 | 20200406194744 | 6 | 2 | 6 | 0 | 15.99 | Hand Creme lotion by Brenda |

| 4 | 2 | 2 | 20200409192334 | 6 | 1 | 6 | 0 | 14.50 | Hair Wash |

| 5 | 2 | 2 | 20200409192334 | 6 | 1 | 6 | 0 | 29.99 | Mini-Cut |

| 6 | 2 | 2 | 20200409192334 | 6 | 2 | 6 | 0 | 15.00 | Nailcare by Brenda |

+---------+---------+--------+----------------+----------+-------------+----------+----------+----------+-----------------------------+

6 rows in set (0.00 sec)

So, after submit -

INSERT INTO TransactionDetails (UserID, TransID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, Nbr2Parm, CharBig )VALUES (1, 3, '20200418154440', 6, 1, 6, 0, 40.0, 'Haircut' )

auto Key was = 7

INSERT INTO TransactionDetails (UserID, TransID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, Nbr2Parm, CharBig )VALUES (1, 3, '20200418154440', 6, 1, 6, 0, 30.0, 'Manicure' )

auto Key was = 8

INSERT INTO TransactionDetails (UserID, TransID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, Nbr2Parm, CharBig )VALUES (1, 3, '20200418154440', 6, 2, 6, 0, 25.0, 'Shampoo by Rosie' )

auto Key was = 9

INSERT INTO TransactionDetails (UserID, TransID, UpdateTime, InfoType, InfoSubType, Validity, Nbr1Parm, Nbr2Parm, CharBig )VALUES (1, 3, '20200418154440', 6, 2, 6, 0, 18.99, 'Super conditioner by Jane' )

auto Key was = 10

SELECT ProdID, UserID, CreateTime, UpdateTime, Pname, Price FROM Product

SELECT ServID, UserID, CreateTime, UpdateTime, Sname, Price FROM Service

Note: The “auto Key was = …" are log info from the App which were captured after the inserts. That key value can be helpful with other INSERT updates.

So, how did those tables get affected? See below -

mysql> select \* from Transaction;

+---------+--------+--------+----------------+----------------+--------+------------+--------+

| TransID | CustID | UserID | CreateTime | UpdateTime | Method | CreditInfo | amount |

+---------+--------+--------+----------------+----------------+--------+------------+--------+

| 1 | 1 | 1 | 20200406194744 | 20200406194744 | 1 | NULL | 60.48 |

| 2 | 2 | 2 | 20200409192334 | 20200409192334 | 1 | NULL | 59.49 |

| 3 | 3 | 1 | 20200418154440 | 20200418154440 | 1 | NULL | 113.99 |

+---------+--------+--------+----------------+----------------+--------+------------+--------+

3 rows in set (0.01 sec)

One can see the new transaction, with a primary key of 3 has been added to the Transaction table. Notice the total amount is $113.99. For the TransactionDetails table -

mysql> select TinfoID,InfoType,InfoSubType,Validity,Nbr2Parm,CharBig from TransactionDetails where TransID = 3;

+---------+----------+-------------+----------+----------+---------------------------+

| TinfoID | InfoType | InfoSubType | Validity | Nbr2Parm | CharBig |

+---------+----------+-------------+----------+----------+---------------------------+

| 7 | 6 | 1 | 6 | 40.00 | Haircut |

| 8 | 6 | 1 | 6 | 30.00 | Manicure |

| 9 | 6 | 2 | 6 | 25.00 | Shampoo by Rosie |

| 10 | 6 | 2 | 6 | 18.99 | Super conditioner by Jane |

+---------+----------+-------------+----------+----------+---------------------------+

4 rows in set (0.01 sec)

We see 4 sub-items added for transaction ID = 3. All record entries have an InfoType of 1. The InfoSubType varies; either 1 or 2, depending on if a service or product.

## Other SQL Queries

Although the Demo App doesn’t exercise nearly all of the possible useful SQL queries on the database, it is certainly possible to manually do them. Who knows, maybe these queries will someday be incorporated into the App. Following are a few interesting queries.

### Equi-join SQL Query

We have described some transaction App processing and shown some of the queries used for that. But perhaps it is useful to use a SQL query that joins the contents of 2 or more tables. For example suppose the user wants all of certain information about a given transaction. A possible query for that is:

mysql> select Transaction.CustID,Transaction.amount,TransactionDetails.InfoType,TransactionDetails.InfoSubType,TransactionDetails.Nbr2Parm,TransactionDetails.CharBig from Transaction,TransactionDetails where Transaction.CustID = 3 and Transaction.TransID = TransactionDetails.TransID;

+--------+--------+----------+-------------+----------+---------------------------+

| CustID | amount | InfoType | InfoSubType | Nbr2Parm | CharBig |

+--------+--------+----------+-------------+----------+---------------------------+

| 3 | 113.99 | 6 | 1 | 40.00 | Haircut |

| 3 | 113.99 | 6 | 1 | 30.00 | Manicure |

| 3 | 113.99 | 6 | 2 | 25.00 | Shampoo by Rosie |

| 3 | 113.99 | 6 | 2 | 18.99 | Super conditioner by Jane |

+--------+--------+----------+-------------+----------+---------------------------+

4 rows in set (0.00 sec)

Here we see a transaction for Customer ID = 3 which had a total cost of $133.99. There were 4 items for that bill; 2 services, haircut and manicure and 2 products, shampoo and conditioner.

This is all well and good. But maybe we don’t care about the Customer ID. We just want to see the customer’s first and last name and the key data for transaction = TransID = 3. No problem:

mysql> SELECT Customer.Fname,Customer.Lname,Transaction.TransID,Transaction.amount,TransactionDetails.InfoType,TransactionDetails.InfoSubType,TransactionDetails.Nbr2Parm,TransactionDetails.CharBig from Customer,Transaction,TransactionDetails where Customer.CustID = Transaction.CustID and Transaction.CustID = 3 and Transaction.TransID = TransactionDetails.TransID;

+-------+-------+---------+--------+----------+-------------+----------+---------------------------+

| Fname | Lname | TransID | amount | InfoType | InfoSubType | Nbr2Parm | CharBig |

+-------+-------+---------+--------+----------+-------------+----------+---------------------------+

| Tammy | Jones | 3 | 113.99 | 6 | 1 | 40.00 | Haircut |

| Tammy | Jones | 3 | 113.99 | 6 | 1 | 30.00 | Manicure |

| Tammy | Jones | 3 | 113.99 | 6 | 2 | 25.00 | Shampoo by Rosie |

| Tammy | Jones | 3 | 113.99 | 6 | 2 | 18.99 | Super conditioner by Jane |

+-------+-------+---------+--------+----------+-------------+----------+---------------------------+

4 rows in set (0.01 sec)

We see Tammy Jones spent $113.99 for 2 services and 2 products. We could expand on that to see the date or other information if we wanted.

### Create a View for SQL Query

It is possible to create a View, which is somewhat akin to another table for the database. As an example, suppose some customers did not pay their bill for services rendered and/or products sold. Using the same privelages as used for creating the table, e.g. root, one can create a view. For example, a view to see which customers have a balance due. The example below was for a host with the database created was “salondb”.

mysql> use salondb;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> CREATE VIEW CustDue\_View AS SELECT Customer.Fname, Customer.Lname,Customer.phone,CustInfo.Nbr2Parm from Customer,CustInfo WHERE CustInfo.Validity <> 0 and CustInfo.Nbr2Parm <>0 and Customer.CustID = CustInfo.CustID;

Query OK, 0 rows affected (0.23 sec)

And then the query from a user login that is allowed access to the salondb -

mysql> SELECT \* from CustDue\_View;

+-------+-------+--------------+----------+

| Fname | Lname | phone | Nbr2Parm |

+-------+-------+--------------+----------+

| Jenn | Snow | 763-294-1300 | 20.00 |

| Tammy | Jones | 612-111-4567 | 113.99 |

+-------+-------+--------------+----------+

2 rows in set (0.01 sec)

This was for a situation described in the database where 2 customers, CustID = 1 and CustID = 3 have a balance due.

mysql> SELECT CustID,InfoType,InfoSubType,Nbr2Parm from CustInfo where InfoType = 5 and InfoSubType = 10 and Validity = 2;

+--------+----------+-------------+----------+

| CustID | InfoType | InfoSubType | Nbr2Parm |

+--------+----------+-------------+----------+

| 1 | 5 | 10 | 20.00 |

| 3 | 5 | 10 | 113.99 |

+--------+----------+-------------+----------+

2 rows in set (0.00 sec)

# Document Comments

**Comments:**

* We met at school on 2/5/2020 to discuss project
* Ask teacher, can we have one or the other either EmpHis table or CustHis table? Or do we need both?
* Should we only allow for walkin appointments? Which would simplify our model, e.g. we wouldn’t probably need an EmployeeSchedule entity.
* Rosie deleted this one the file Feb,05 2020
  + Chairs can also be scheduled as unavailable by either a manager or receptionist for several reasons; there is no stylist available that day for that chair, a stylist is away for lunch or other reason, etc.
  + \*Chairs are scheduled on ½ hour intervals per the entire yearly schedule of the business. But they can be scheduled for repair and thus out of service for that period of time. When scheduled for repair all the no-longer available time slots are scheduled to a manager and that manager is responsible for putting that chair back into the availability rotation when the repair is completed.
  + \*sChairs are scheduled to customer where a stylist can provide a service, such as shampoo, haircut, manicure, and/or hair-dying.
  1. Chair to ChairHist is one to many,
  2. Chair to ChairSchedule is one to one,
  3. ChairSchedule to Employee is one to zero or one,
  4. Service to ChairSchedule can be many to one (a chair and time slot can be for multiple services, e.g. haircut and also a shampoo,
* Roise deleted this on Feb 19th(no need for chair assignment)
  + \*Chair (ChairID, CreatedTimeDate, LastUpdatedTimeDate, AvailableIndicator, PhysicalLocation)
  + \*ChairHistory (ChairHistID, CustID, UpdaterEmpID, ActivateIndicator, LastUpdatedTimeDate)
  + \*ChairSchedule (ChairID, UpdaterEmpID, EmpID TimeSlot, AvailableIndicator, ServiceID)
  + Custodial work will also be contracted out. None of the salon workers will be employed to do cleaning, etc. except as part of the process of cleaning around the salon chairs or whatever.
  + subcontractor stylist who lease the use of a salon chair for certain periods of time
  + options to expand within their leased space. Therefor the salon could be as small as one salon chair to as much as 20 salon chairs or any amount between 1 and 20.
  + The business model is that for every salon chair available on a given day a stylist is assigned to or has leased the use of that chair.
  + EmployeeSchedule (EmpID, UpdaterEmpID, TimeSlot, AvailableIndicator)Employee to EmployeeSchedule is one to one,