# CS374 – Database Management

# Database Final Project

# Rubric for Final Project

## Group Member #1: Connor Weldy

## Group Member #2: Nozomu Ohno

## Group Member #3: Michael Laramie

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Requirements | Points | Awarded |
| Database Design | * ER/UML diagram included * Database conforms to ER/UML diagram * Keys and Foreign Keys are defined appropriately * Database redundancies are eliminated or documented | 30 |  |
| Queries | * Queries execute correctly * Queries in English included * Queries satisfy the client requirements | 30 |  |
| Application | * Code found easily  (This is a department assessment project) * Application works * Embedded SQL written as appropriate * Code well thought out, well commented * Application is well tested – that is, it is not easy to break | 40 |  |

**Application Code Design**

Files:

main.py - contains code to run our application. Contains a variety of classes, that act as screens for the user to interact with

sqlfunctions.py - contains functions that execute inputted SQL statements

userinterface files: account.ui, accountdetails.ui, accountgoals.ui, accountMonth.ui, deleteTransaction.ui, login.ui, newAccount.ui, newTransaction.ui, newuser.ui, welcomescreen.ui

Requirements to run the code:

* Python 3.8
* Userinterface files listed above in subfolder /userinterfaces/
* Sqlfunction.py and main.py
* PyQt5 library (need to pip install PyQt5)
* DBMS: SQl server
* Whitworth University Internet Connection

Introduction:

For our application we utilized Python programming lanaguage along with PyQt5 user interface library. We designed user interfaces using Qt Designer GUI which allows developers to easily create user interfaces. Qt Designer creates the user interfaces as .ui files, in xml code.

PyQt5 is utilized within the python code to access, interact with, delete, and create the user interface elements. Qt utilizes objects called widgets which have several subclasses. Objects such as buttons, line edits, and labels are part of the Qt widgets.

In our code, we developed an application for personal financial management. We tried to accelerate the user interface design by using Qt Designer so we could focus more on the actual code and database interactions.

Code Layout:

Our python script contains several classes that act as screen displays in our application. Each screen class has its own .ui file with a variety of widgets such as buttons, labels, tables, drop down menus, and more.

Each class also contains several methods that allow users to jump from screen to screen.

The backbone structure of our application is called a QStackedWidget. This Qt built in widget acts as a stack for user interfaces. We first start by adding our welcome screen class/widget to the stack. When we want to go a different page, we add an instance of that specific screen class to the widget, and then set the index of the stacked widget to the respective instance.

We also utilize “mini applications” within some of our classes. We create a whole separate stacked widget for a specific screen to allow the use of pop-up screens.

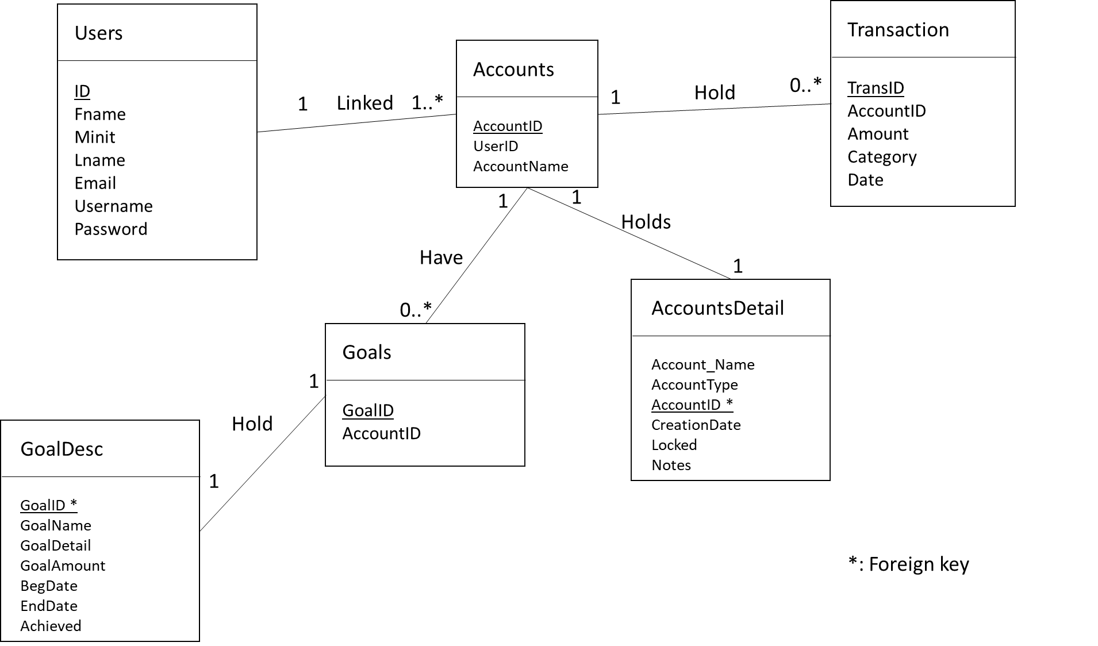
In our main function, as soon as the backbone QStackedWidget is set to .show(), then the application is running, and the user can interact with the various aspects and screens of the application.

SQL Queries:

Throughout our code we run specific SQL queries to display various information to the user. For example, on our account details page, we have several queries that are built within the constructor that are specific to the accounts detail page. They retrieve information such as username, account name, account transactions, current account total, and more.

We also have a sqlfunctions.py file that contains functionalized code for executing the queries.

**UML Diagram:**

**Description:**

In our UML design above, we have six tables. The first table on the left named “Users” contains all user information. Each user has a unique ID, a first name (Fname), a last name (lname), an email (Email), a username (Username) and a password (Password). The second table named “Accounts” contains information about an account. Each account has a unique ID (AccountID), a user ID which is a foreign key to the Users table (UserID) and an account name (AccountName). The third table called “Transactions” holds all information about a transaction. It will contain a unique transaction ID (TransID), an account ID (AccountID) which is a foreign key to the Accounts table (AccountID), a transaction amount (Amount), a category (Category), and the date of the transaction (Date). The fourth table named “AccountsDetail” contains detailed information about an account. Each account detail has a foreign key (AccountID), account name (AccountName), account type (AccountType), creation date, if the account is locked (Locked) and notes. The fifth table named “Goals” contains information about a goal set by user that is connected to an account. Each goal has a goal ID and an AccountID associated with it. The sixth table named “GoalDesc” contains description of a goal. Each goal description has a foreign key (GoalID), goal name (AccountName), more details of the goal, goal amount set by the user, date the goal starts, date the goal ends, and a check whether the goal is achieved.

Each user in our database is linked to one or many accounts. Accounts have only one user. Accounts hold several transactions, but transactions only have one account. One account can only hold one account detail, and one detail can only describe one account. One account can have many goals, but one goal can be associated with only one account. One goal can only hold one goal description, and one description can only describe one goal.

**Queries: We have a lot.** Anything in \* \* denotes current user variables that needed to be appended to the query.

1. SELECT Username FROM Users WHERE Username = \*username\*
   1. Retrieve the username from users that matches the username entered by the users.
2. SELECT Max(ID) as userid FROM Users
   1. Retrieve the highest user ID that is being used right now from the users table.
3. INSERT INTO Users (ID, Fname, Minit, Lname, Email, Username, Password) VALUES(\* userID, firstName, middleInital, lastName, email\* + "', '" + username + "', '" + password + "');
   1. Add a new user with unique user ID, entered name, email address, username and password.
4. SELECT \* FROM Users WHERE Username = \*username\*
   1. Retrieve all information about a single user that is chosen by the user.
5. SELECT ID From Users WHERE Username = \*username\*
   1. Retrieve the user ID of one user that the user entered.
6. SELECT Max(AccountID) as accountId FROM Accounts
   1. Retrieve the highest account ID that is being used right now from the accounts table.
7. SELECT AccountDetail.Account\_Name FROM Accounts, AccountDetail WHERE Accounts.UserID = \*UserID\* AND Accounts.AccountID = AccountDetail.AccountID AND AccountDetail.Account\_Name = \*accountName\*
   1. Retrieve the account name from accounts table that has the account ID and the account name entered by the user.
8. INSERT INTO AccountDetail VALUES(\*AccountName, AccountType, accountId, creationDate, 0, notes\*)
   1. Add a new account detail with the user entered account name, account type, account ID, and the date created with false for if the account is locked.
9. INSERT INTO Accounts VALUES(\*accountId, userId\*)
   1. Add a new account with a new account ID and the user ID of the user that is creating the account.
10. SELECT Username FROM Users WHERE ID = \*userId\*
    1. Retrieve the username from the user table that has the specified user ID attached to it.
11. SELECT AccountDetail.Account\_Name FROM AccountDetail, Accounts WHERE Accounts.UserID = \*userId\* AND Accounts.AccountID = AccountDetail.AccountID
    1. Retrieve the account name from account details table that has the account ID and the account name entered by the user.
12. SELECT AccountID FROM AccountDetail WHERE Account\_Name = \*accountName\*
    1. Retrieve the account name from account details table that has the account ID and the account name entered by the user.
13. SELECT Max(TransID) as transId FROM Transactions
    1. Retrieve the highest transaction ID that is being used right now from the transactions table.
14. INSERT INTO Transactions VALUES(\*transId, AccountId, transAmount, transCategory, transDate\*)
    1. Add a new Transaction with a new transaction ID, the account ID of the account associated with the transaction. amount, category and the date of transaction
15. SELECT TransID FROM Transactions WHERE TransID = \*transID\*
    1. Retrieve the transaction ID from the transaction table that is specified by the user.
16. DELETE FROM Transactions WHERE TransID = \*transId\*
    1. Remove a transaction that have the same transaction ID as the one entered by the user from the transactions table.
17. SELECT Account\_Name FROM AccountDetail WHERE AccountID = \*AccountID\*
    1. Retrieve the account name from the account details table that has the account ID entered by the user.
18. SELECT sum(amount) as total FROM Transactions WHERE AccountID = \*AccountID\*
    1. Retrieve the sum of transaction amount from the transactions table that belongs to the account ID specified by the user.
19. SELECT CreationDate FROM AccountDetail WHERE AccountID = \*AccountID\*
    1. Retrieve the date account was created from account details table that is associated with the account ID entered by the user.
20. SELECT \* FROM Transactions WHERE AccountID = \*AccountID\*
    1. Retrieve all transactions from transactions table that is associated with the account ID entered by the user.
21. SELECT Account\_Name FROM AccountDetail WHERE AccountID = \*AccountID\*
    1. Retrieve account name from account details that is associated with the account ID entered by the user.
22. SELECT sum(amount) as monthlyDeposit FROM Transactions WHERE Amount > 0 AND AccountID = \*AccountID\* AND Date <= (SELECT EOMONTH(\*DateMonth\*) AND Date >= (SELECT DATEADD(Day, 1, EOMONTH(\*DateMonth-1\*)
    1. Retrieve the sum of all deposits from transactions table that is in the month that user specified and is associated with the account ID entered by the user.
23. SELECT sum(Amount) as monthlySpent FROM Transactions WHERE AMOUNT < 0 AND AccountID = \*AccountID\* AND Date <= (SELECT EOMONTH(\*DateMonth\*) AND Date >= (SELECT DATEADD(Day, 1, EOMONTH(\*DateMonth-1\*)
    1. Retrieve the sum of all withdrawals from transactions table that is in the month that user specified and is associated with the account ID entered by the user.
24. SELECT Category, SUM(ABS(Amount)) AS total FROM Transactions WHERE Amont < 0 AND AccountID = \*AccountID\* AND Date <= (SELECT EOMONTH(\*DateMonth\*) AND Date >= (SELECT DATEADD(Day, 1, EOMONTH(\*DateMonth-1) GROUP BY Category ORDER BY total desc
    1. Retrieve the sum of all withdrawals by category from transactions table that is in the month that user specified and is associated with the account ID entered by the user and order it by the highest amount to lowest.
25. SELECT TOP 1 Category FROM (SELECT Category, SUM(ABS(Amount)) AS total FROM Transactions WHERE AccountID = \*AccountID\* AND Date <= (SELECT EOMONTH(\*DateMonth\*) AND Date >= (SELECT DATEADD(Day, 1, EOMONTH(\*DateMonth-1\*)) GROUP BY Category HAVING sum(amount) < 0) AS t ORDER BY total desc
    1. Retrieve the most spent category from the table of sum of spendings grouped by category from a specified month and account ID specified by the users.