CommuniTools Report

CSCI 370 Final Project

Instructor: Liu Huizhu
By: Jasper Charlinski
Due: April 18th

Project Proposal:

Introduction:

This proposal outlines the database application CommuniTools. The purpose of this application is to allow members of a community to share tools with one another, without the need for payment. The application aims to facilitate the borrowing and lending of tools, making it easier for community members to complete DIY projects, repairs, and maintenance.

The CommuniTools application is designed to promote collaboration and communication within communities by making tool sharing more accessible and convenient. The application is targeted towards small communities where members are interested in sharing tools with one another. With this tool sharing application, community members can borrow the tools they need for a specific amount of time without incurring additional costs.

The CommuniTools application is developed using C++ and Oracle's OCCI library, iostream, termios, unistd, cstdlib, and iomanip.

Audience:

The target audience for the CommuniTools application is members of specific small communities who are interested in sharing tools with one another. The application is designed to appeal to individuals who are community-minded, open to meeting new people, and interested in DIY projects, repairs, and maintenance. CommuniTools is built for users who are comfortable with command line interfaces and enter honest information.

The tool sharing application is designed to cater to the needs of people who might not have the resources to purchase their own tools, or who might not have access to the tools they need for their projects.

Functionalities:

The following requirements have been identified from the hypothetical user:

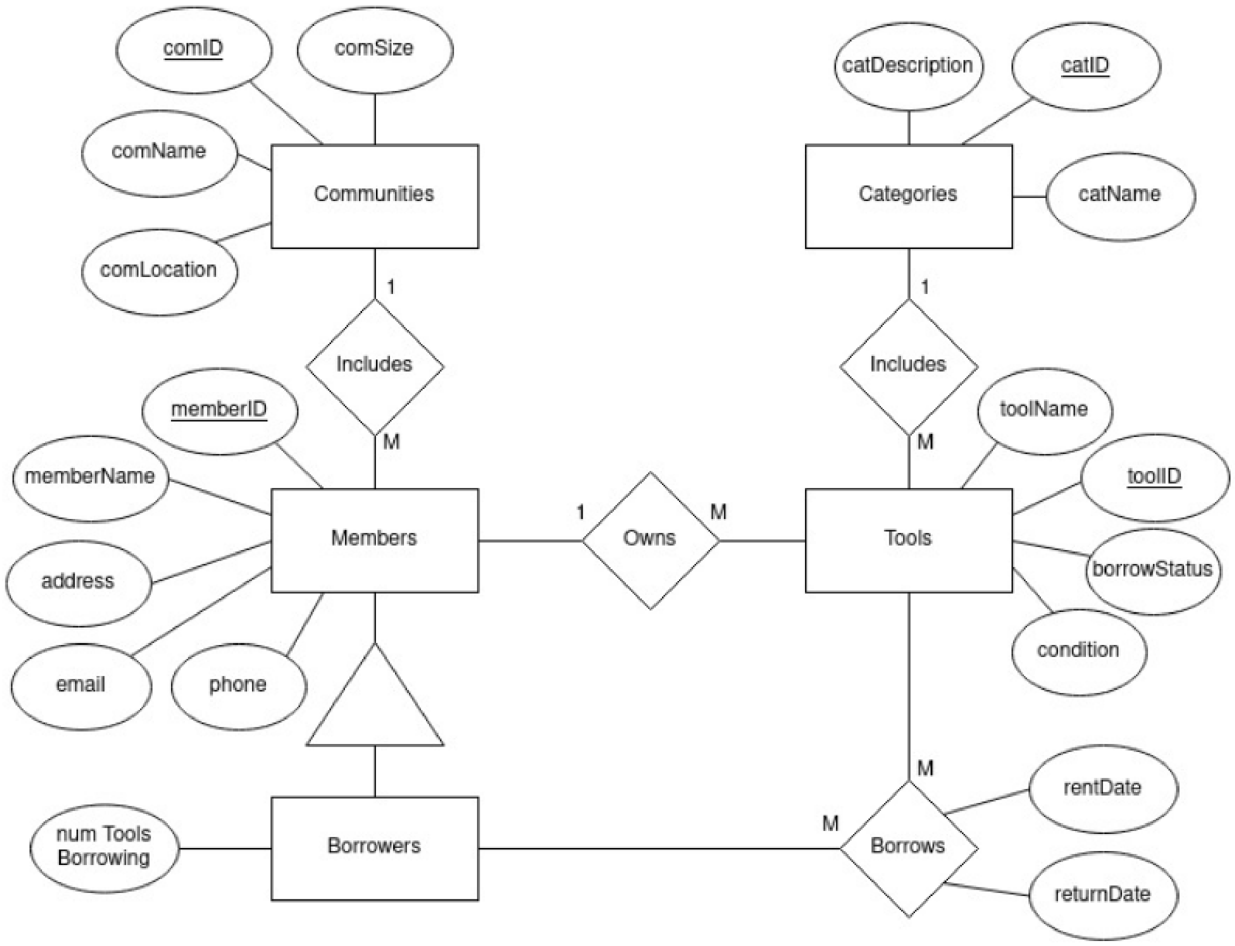
1. **Registration:** Users can create a new community member account. The registration process requires users to provide a referral member by ID, their community ID, their name, address, and contact information.
2. **Tool Listing:** Users can list the tools they own and are willing to lend to others. Each tool listed will be associated with one of the predetermined categories.
3. **Tool View:** Users can view all tools in the database, all tools in a given category, and all tools that they can currently borrow.
4. **Tool Borrowing:** Users can borrow available tools within their community. Users are limited on the number of tools they can borrow at one time

5. **Tool Return:** Users can return a tool they are currently borrowing.
6. **Tool Removal:** Users can remove a tool if they no longer want to lend it out.
7. **Login verification:** Users must have valid login credentials to the Oracle database as well as the application.

Constraints:

- **Character Limit:** Users are limited to a character limit on the input data which is specified in the table descriptions.
- **Security and Validation:**
  - User input is not sanitized however all statements are prepared in construction of the program which prevents SQL injections.
  - Account passwords are stored in plain text which is also a security risk.
  - User input such as email, phone, and address is not verified to be in the correct format or a valid value.

ER Diagram:



Schema:

Communities (comID, comName, comLocation, comSize)

CommunityMembers (memberID, comID, memberName, address, email, phone, username, password)

Borrowers (memberID, numToolsBorrowing)

CommunityTools (toolID, catID, memberID, toolName, borrowStatus, condition)

ToolCategories (catID, catName, catDescription)

BorrowRecord (recordID, memberID, toolID, rentDate, returnDate)

Table descriptions:

Communities:

```
comID NUMBER(5) PRIMARY KEY ,
comName VARCHAR(256),
comLocation VARCHAR(256),
comSize NUMBER (8, 0)
```

CommunityMembers:

```
memberID NUMBER(5) PRIMARY KEY,
comID NUMBER(5) REFERENCES Communities,
firstName VARCHAR(256),
lastName VARCHAR(256),
address VARCHAR(256),
email VARCHAR(64),
phone VARCHAR(15),
username VARCHAR(32)
password VARCHAR(32)
```

Borrowers:

```
memberID NUMBER(5) PRIMARY KEY REFERENCES CommunityMembers,
numToolsBorrowing NUMBER(2, 0) DEFAULT 0
```

BorrowRecords:

```
recordID NUMBER(5) PRIMARY KEY,
memberID NUMBER(5) REFERENCES CommunityMembers,
toolID NUMBER(5) REFERENCES CommunityTools,
rentDate DATE,
returnDate DATE
```

ToolCategories:

```
catID NUMBER(5) PRIMARY KEY,
catName VARCHAR(256),
catDescription VARCHAR(512)
```

CommunityTools:

```
toolID NUMBER(5) PRIMARY KEY,
catID NUMBER(5) REFERENCES ToolCategories,
memberID NUMBER(5) REFERENCES CommunityMembers,
toolName VARCHAR(256),
borrowStatus NUMBER(1, 0) DEFAULT 0,
condition VARCHAR(256)
```

Database interaction:

The CommunityMembers and CommunityTools tables interact with user input when creating an account, adding or deleting a tool, or displaying available tools.

The Borrowers and BorrowRecords tables are updated by the system when the user borrows a tool and do not directly deal with user input.

Communities and ToolCategories tables remain static and are never modified. Only the DBM can create or update entries in these tables.

Code Documentation:

How to Run:

```
Enter sql/ folder.
Run sqlplus
Enter @create_database.sql then @insert.sql

Return to CommuniTools/ folder and run makefile to compile.

Run program executable: ./communiToolsx

Enter database login.

Enter application login,

Test login:
username: user
password: pass

If login was successful, the option menu will be shown.
```

Testing:

Test data:

Test data is located in sql/insert.sql
It contains inserts for 10 communities, 16 members, 10 categories, and 18 tools. No records are inserted in this file as the application handles the logic behind inserting a record and updating the necessary table, however record inserts were tested throughout the development of the project. The insert statements for members, categories, and tools were generated by ChatGTP3 and were altered slightly to make them consistent with the application.

Database connection testing:

Test 1 - connection failed:
Case: Connection to Oracle database failed.
Expected output:

```
ORA-#: Some connection error
<program ends>
```

Login testing:

Test 1 - Invalid database login:
Case: When prompted for database login invalid login credentials are provided.
Expected output:

```
ORA-01017: invalid username/password; logon denied
```

Test 2 - Invalid application login:
Case: When prompted for application login invalid login credentials are provided.
Expected output:

```
<invalid login>
Invalid username or password try again.
Number of tries remaining: 2

<invalid login>
Invalid username or password try again.
Number of tries remaining: 1

<invalid login>
Invalid username or password try again.
Number of tries remaining: 0

Out of login attempts, quitting program…
<program ends>
```

Test 3 - Valid login:
Case: valid credentials are provided for both database and application logins when prompted.

Expected output:
<database login>
---------- Database Connection Successful ----------

<application login>
------------------- Login Successful -------------------
<shows menu>

ID validation testing:

Test 1 - Invalid ID provided:
Case: User enters an invalid ID when prompted to enter an ID of tuple in a specified table.
Expected output:
Invalid <table> ID. <method returns false>

Add new member testing:

Test 1 - Member already exists with same name:
Case: User provides a valid reference member and community ID but a member already exists in the database with the provided name.
Expected output:
<firstName> <lastName> is already a member.

Test 2 - Insert into database failed:
Case: User provides valid information for the new member but the database fails to insert entry into the CommunityMembers table.
Expected output:
Error: Failed to update record
Failed to add member.

Test 3 - Insert into database successful:
Case: User provides valid information for the new member and entry is successfully inserted into the CommunityMembers table.
Expected output:
Added member <firstName> <lastName> with user name: username successfully. You can now log in as <username> with your set password.

<New entry added to CommunityMembers table>

Add new tool testing:

Test 1 - Insert into database failed:
Case: User provides valid information for the new tool but the database fails to insert entry into the CommunityTools table.
Expected output:

Error: Failed to update record
Failed to add tool.

Test 2 - Insert into database successful:
Case: User provides valid information for the new tool and entry is successfully inserted into the CommunityTools table.
Expected output:
Added tool <tool name> successfully.
New entry added to CommunityTools table

Show all tools testing:

Test 1 - All tools selected:
Case: User enters a valid ID of a category.
Expected output:
Prints <toolName>, <firstName>, <lastName>, <comName>, <borrowStatus>, <condition>
From selected option

Borrow tool testing:

Test 1 - Tools is already being borrowed:
Case: User enters a valid ID of a tool but the tool is already being borrowed.
Expected Output:
Tool <toolID> is already being borrowed.
Failed to borrow tool.

Test 2 - User is already borrowing the maximum amount of tools:
Case: User enters a valid ID of a tool and it is available to be borrowed but the user is currently borrowing 5 tools.
Expected output:
You are currently borrowing <numToolsBorrowing> tools, which is the maximum amount of borrows at one time, return some tools before borrowing more.
Failed to borrow tool.

Test 3 - Insert into database failed:
Case: User provides valid information to borrow tool but the database fails to insert entry into the BorrowRecords table or Borrowers table or updated numToolsBorrowing if user is already in Borrowers table.
Expected output:
Error: Failed to update record
Failed to borrow tool.

Test 4 - Insert into database successful:

Case: User provides valid information to borrow a tool and entry is successfully inserted into the BorrowRecords table and Borrowers table or updated numToolsBorrowing if the user is already in Borrowers table.
Expected output:
You are now borrowing <toolName> from <memberName> ID: <memberID>
you can pick up your tool at <address> and contact them at Email: <email> or Phone: <phone>

<CommunityTools borrowStatus value is updated, new entry is added to BorrowRecords table, new entry is added to Borrowers table or numToolsBorrowing is updated if user has already borrowed a tool.>

Return tool testing:

Test 1 - User is not borrowing tool:
Case: User enters a valid tool ID but there is no record of the current user borrowing the tool where the return date is null.
Expect output:
You are not currently renting the tool with ID <toolID>.
Failed to return tool.

Test 2 - Tool is returned successfully:
Case: User enters a valid tool ID of a tool they are currently borrowing.
Expected output:
Tool has been returned.

<BorrowRecords returnDate is updated to SYSDATE, CommunityTools borrowStatus is updated, Borrowers numToolsBorrowing is updated.>

Remove tool testing:

Test 1 - User does not own tool:
Case: User enters a valid ID of a tool but they do not own the tool.
Expected output:
You do not own this tool.
Failed to remove tool.

Test 2 - Removal from database failed:
Case: User enters a valid ID of a tool that they own, but database failed to delete entry.
Expected output:
Error: Failed to update record.
Failed to remove tool.

Test 3 - Removal from database successful:
Case: User enters a valid ID of a tool that they own and tool is deleted from database.

Expected output:
Tool has been removed successfully.

<Entry is removed from CommunityTools table and toolID is set to null for all BorrowRecords containing that tool>

Code Description:

SQl and Database:

The tables needed for the program to run are created with the create_database.sql file and testing data is inserted with insert.sql.

Each primary key in the database is set to auto increment starting from one, this way the application does not have to worry about creating unique IDs. However the security risk with doing this is acknowledged, as it is easy for the user to know other users IDs and therefore easy to impersonate other members. .

Another security risk to mention is that the passwords of the users are stored in plain text. Obviously if this was a public application this would be a bad idea, however for the purposes of this project I chose to save time by not hashing the passwords.

The environment and connection to the database is established in the constructor of the Database class, and is then terminated in the destructor.

All SQL statements are prepared in the constructor of the CommuniTools class as private attributes. Each method in the class then has access to execute any of the queries. The statements are then terminated in the destructor.

Classes:

Database:

The Database class is responsible for creating the Oracle environment and attempting to connect to the database with a given username, password, and connect string. This class acts as the database controller for the application. The constructor initializes the Oracle environment and connects to the database. It throws an exception if the connection fails, and the application terminates. The destructor terminates the environment and connection.
The Database class is also responsible for validating primary keys of a table. Since the primary keys for each table are set up to auto increment from 1, the largest primary key will be the number of tuples in that table. Therefore primary keys can be checked by seeing if it is less than 1 or greater than COUNT(*) FROM table.

CommuniTools:

The CommuniTools class is responsible for controlling the business layer of the application. It creates a private instance of the Database class and initializes all of the prepared statements used in the application in the constructor. It terminates all of the statements in the destructor. This way, each statement used in the application is only created once and SQL injections are prevented.

The CommuniTools class has methods for each functionality, including addMember, addTool, showTools, borrowTool, returnTool, and unlistTool. As well as helper methods to display information including showUsersTools, showAvailableTools, showToolBorrows. Each handles its specific functionality and interacts with the database through the prepared statements. This class also contains methods to display the option menu and get user commands to call the respective method as well as the method to validate the users login credentials for the application.

Main:

The main is responsible for initializing the CommuniTools object CT when the program is run. The user is first prompted to enter their Oracle login and if valid their application login by calling CT.logIn. If both were successful then the application loop is entered and the user will be shown the option menu by calling CT.printMenu and executing the command with CT.getOption.

Makefile:

Complied with: g++-4.9 -std=c++11
Using flags:
-I/usr/local/lib/Oracle/instantclient_11_2/sdk/include -Wall -Wextra
-L/usr/local/lib/Oracle/instantclient_11_2 -locci -lociei

To executable: communiToolx

Conclusion and Future Work:

In the end I am happy with the result of this project, I completed all of the functionalities that I set out to do at the beginning of this project.

Not everything went smoothly however, due to the cryptic nature of the OCCI error messages there were a few times where a relatively simple bug was difficult to pinpoint and lead to tedious testing. But after becoming familiar with OCCI I can spot the errors in the code much faster as I now know what to look for. Also the use of GDB helped tremendously as it gave a line number where the error was occurring.

About a third of the way through implementation I decided to make all of the statements be prepared in the constructor, this was initially a challenge to figure out but once I got it working it was the best decision I made in the project. It cleaned up my code drastically and led to better efficiency and security.

If I were to continue with this project, the next feature I would add is increased security. I would like to implement hashed passwords and more strict verification for user input. For example using regular expressions to verify provided emails and phone numbers.