Ian Eliopoulos(10067211) , Mike Simister (10095107), Brian Yee (00993104), Oleksiy Brylov (10123597)

F2015

CPSC471 pROJECT

Tennis Club Database

**Table Of Contents**

**Abstract**……………………………………………………. 1

**I: INTRODUCTION**……………………………………… 2

**II: PROJECT DESIGN**…………………………………… 3

Users and Transactions……………………………….. 3

ER Diagram V1…………………………………..,…... 5

ER Diagram V2……………………………………..… 6

List of changes…………….………………………….. 7

**III: IMPLEMENTATION**……………………………….... 9

Relational Schema Diagram…………………………… 9

Database Management System (DBMS) Selection.......... 12

SQL Statements………………………………………… 13

**IV: USER INTERFACE...**…………………………………… 20

**V: CONCLUSION**…………………………………………… 22

**ABSTRACT**

The idea for this project came from a group member who spends the summer playing tennis at a local tennis club. This particular group member has often thought about the prospect of modernizing some of the outdated processes of the club. In fact, during one conversation with the club president, the idea to implement an electronic database came up.

It was decided that this database, if it were to be implemented, should at a minimum:

* Be stored in one and only one location
* Be secure and remotely accessible
* Provide varying levels of access for:
  + Members
  + Board members
  + admin staff
  + employees
* Have the ability to track typical daily transactions such as:
  + Guest Fee Purchases
  + Tennis Ball Purchases
  + Event Fee Purchases
  + Event Signup

The group member and the club president were proponents of change. They would both like to see their club improve and eliminate some of the inefficient processes. Once the group member learned about the CPSC471 term project for the fall semester, it was a natural choice to propose the tennis club database as a possibility. The other members of the group were on board with the idea. All of the members were excited by the prospect of working on a real world solution.

**I: INTRODUCTION**

**The Problem**

A local tennis club needs a database to keep track of members, board members, senior members, employees, events, tournaments, and equipment/inventory. The club currently tracks this information on paper or possibly on individually maintained Excel spreadsheets. The club currently has a website which could possibly be used for remote access to the database by way of a web application. The website is hosted by a local company. As the club has not formally asked for a database or web application, this project would be based on our opinion of the club’s expected requirements. However, no actual personal/sensitive information about the club or its members will be used. The result of this project could be used by this or any similar club with similar requirements.

**The Solution**

The solution has been designed as a web application. It initially has different levels functionality for 3 types of users. Those users being: member, employee, administrator (board member). The interface requires a password for initial login. After the user logs in, they are able to various tables of information depending on the type of user they are. More details on the users and their abilities is found in the project design section.

The application is accessible from remote hosts and could be easily served from an existing web server or from a new server dedicated to hosting only the database. In either case a link on the current club website could be used as a login portal.

The most complicated relation in the database is the member relation, after all a club’s focus should be on its members. For this reason the application is built around members and those relations that are most important to the club and its members. At this point, not all relations are accessible through the UI. However since there are functions which support Create, Retrieve, Update, Delete (CRUD) as standard operations it is somewhat trivial to attach this functionality to the additional relations.

Some of the interface uses Bootstrap components for input boxes, drop down lists and girds while other pages utilize standard HTLM components. It would be up to a particular end user to decide on these or perhaps other standard tools in order to have a consistent look and feel across all pages of the UI.

This system meets the expectations originally proposed and with the addition of general functions to handle CRUD operations it even exceeds them.

**II: Project Design**

**Users and Transactions**

There are three types of users of the system: club members, employees, and administrators. Each user expands upon the permissions of the previous user type. Below is the complete transaction collection for each user type divided by the corresponding type of SQL query.

1. **Club Member**

Club members are the simplest type of user. They are limited to basic event interactions. Club member transactions are as follows:

**Retrieve**

* View the dates and times of upcoming events

**Update**

* Sign up as a volunteer for an event
* Sign up to participate in an event (i.e. tournaments)

1. **Employee**

An employee user is primarily responsible for sales. All employees are club members. As a result, they have the same permissions as regular club members but also have abilities pertaining to sales and member information. Employee transactions are as follows:

**Retrieve**

* View the dates and times of upcoming events
* View the list of club members
* Viewing the list of inventory items
* Viewing the list of sales transactions

**Update**

* Sign up as a volunteer for an event
* Sign up to participate in an event (i.e. tournaments)
* Updating inventory information

**Create**

* Adding a sales transactions
* Adding a new item to the inventory list

1. **Administrator**

Administrator is the most advanced user of the system. They have the highest level of permissions, and they are able to perform every transaction the system interface is capable of. Administrators would be high ranking employees, or board members, etc. It should be noted that we make a distinction between administrators and *system* administrators. System administrators would require actual database knowledge, and would thus be capable of understanding and editing the database through a DBMS. System administrators would have access to the database itself and would therefore also have the same permissions as an administrator. Administrator transactions are as follows:

**Retrieve**

* View the dates and times of upcoming events
* View the list of club members
* View the list of inventory items
* View the list of sales transactions

**Update**

* Sign up as a volunteer for an event
* Sign up to participate in an event (i.e. tournaments)
* Update inventory information
* Update member information

**Create**

* Add sales transactions
* Add a new item to the inventory list
* Add new members





**ER Diagram - List of Changes**

1. Entity: **Administrato**r
   1. Created Entity type
   2. Created Attribute id
   3. Created relationship “is”
2. Entity: **Department**
   1. Identified the primary key
3. Entity: **Employee**
   1. Added an attribute: “password”
   2. Added a 1:1 relationship : “works as”
4. Entity: **Event** 
   1. Added an attribute “id” as the new primary key
5. Entity: **Guardian**
   1. Changed firstname attribute to a partial key
   2. Changed middlename attribute from multivalued to single valued
6. Entity: **Inventory Items**
   1. Added underline to the primary key inventorynum
7. Entity: **Member**
   1. Changed middlename attribute from multivalued to single valued
   2. Added attribute “password”
   3. Added attribute “active”
8. Entity: **Pro**
   1. Changed middlename attribute from multivalued to single valued
9. Entitry: **Boardmember**
   1. Changed cardinality from 2:N to M:N in the “hires” relationship
10. Entity:**Social Event**
    1. Changed attribute guest to membersonly
11. Entity: **Transaction**
    1. Created Entity type
    2. Created Attribute transactionid (primary key)
    3. Created Attributes date, amount, totals
    4. Created Relationships “purchased”, “creates”, “sold”
12. Entity: **Interclub Team**
    1. Changed Cardinality from 1:12 to 1:N

**ER Diagram - List of Entities and Private Keys**

Administrator – id  
Board member – positionType  
Member – memberID  
Pro – email  
Junior member – memberID  
Guardian – name  
Interclub team – level  
Event – eventid  
Social event – eventid  
Tournament event – eventid  
Department – number  
Inventory Items – inventorynum  
Employee – id  
Transaction – transactionid

**List of Relationships, Participating Entities, and Cardinality**

Hires – Board member, Pro – M:N  
Manages – Board member, Department – N:1  
Serves as – Board member, Member – 1:1  
Is – Administrator, Member – 1:1  
Gives lessons – Pro, Member – M:N  
Participates in – Member, Event – M:N  
Volunteers for – Member, Event – M:N  
Plays for – Member, Interclub Team – N:M  
Purchased – Member, Transaction – 1:N  
Works as – Member, Employee – 1:1  
Looks after – Guardian, Junior member – 1:1  
Runs – Event, Department – N:1  
Sells – Department, Inventory items – M:N  
Works for – Department, Employees – 1:N  
Items sold – Inventory items, Transaction – 1:N  
Cashier id – Transaction, Employee – N:1  
Supervises – Employee, Employee – 1:N

**III: IMPLEMENTATION**

**Relational Schema Diagram**

The algorithm to convert the ER diagram to the relational schema diagram used was the algorithm discussed in class (summarized below). The steps are color coded so that in the relational schema diagram it is clear which step was applied to obtain which part of the relational schema diagram.

**Step 1: Mapping of Regular Entity Types**

**Step 2: Mapping of Weak Entity Types**

**Step 3: Mapping of Binary 1:1 Relationship Types**  
**Step 4: Mapping of Binary 1:N Relationship Types**

**Step 5: Mapping of Binary M:N Relationship Types**

**Step 6: Mapping of Multivalued Attributes**

**Step 7: Mapping of N-ary Relationship Types**

Step 8: Options for Mapping Specialization or Generalization.

Step 9: Mapping of Union Types (Categories).

**Administrator**

|  |  |
| --- | --- |
| ID | MEMBERID |

FK

**Boardmember**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POSITIONTYPE | EMAIL | POSHELDSINCE | DEPTMANAGED | MEMBERID |

FK FK

**Department**

|  |  |
| --- | --- |
| NUMBER | NAME |

**Employee**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | ROLE | PHONENUM | EMAIL | STARTDATE | HOURS | WAGE |

**Employee (Continued)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FIRSTNAME | MIDDLENAME | LASTNAME | STREETNUM | POSTALCODE | STREETNAME |

**Employee (Continued)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CITY | SUPERVISORID | DEPT | PASSWORD | MEMBERID |

FK FK

**Event**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | NAME | DATE | MINLIMIT | MAXLIMIT | BUDGET | FEE | DEPT |

FK

**Gives Lessons**

|  |  |
| --- | --- |
| PROEMAIL | MEMBERID |

FK FK

**Guardian**

|  |  |  |  |
| --- | --- | --- | --- |
| FIRSTNAME | MIDDLENAME | LASTNAME | STREETNUM |

**Guardian (Continued)**

|  |  |  |  |
| --- | --- | --- | --- |
| STREETNAME | POSTALCODE | CITY | JUNIORID |

FK

**Interclub Team**

|  |  |  |
| --- | --- | --- |
| LEVEL | CAPTAINID | CO-CAPTAINID |

**Inventory\_Items**

|  |  |  |  |
| --- | --- | --- | --- |
| INVENTORYNUM | NAME | PRICE | QUANTITY |

**Juniormember**

|  |  |
| --- | --- |
| BIRTHDATE | MEMBERID |

FK

**Member**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MEMBERID | RATING | PHONENUMBER | FIRSTNAME | MIDDLENAME | LASTNAME |

**Member (Continued)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RENEWALDATE | STREETNAME | STREETNUM | DATEJOINED | POSTALCODE |

**Member (Continued)**

|  |  |  |
| --- | --- | --- |
| CITY | PASSWORD | ACTIVE |

**Participates\_in**

|  |  |
| --- | --- |
| MEMBERID | EVENTID |

FK FK

**Pro**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMAIL | RATE | PHONENUMBER | FIRSTNAME | LASTNAME | MIDDLENAME | HIREDBY1 |

FK

**Sells**

|  |  |
| --- | --- |
| DEPTNUMBER | INVENTORYNUM |

FK FK

**Social\_event**

|  |  |
| --- | --- |
| EVENTID | MEMBERSONLY |

FK

**Tournament**

|  |  |  |  |
| --- | --- | --- | --- |
| EVENTID | WINNERID | PRIZE | DRAW |

FK

**Transaction**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TRANSACTIONID | DATE | MEMBERID | INVENTORYNUM | AMOUNT | TOTALS | CASHIERID |

FK FK

**Volunteers\_for**

|  |  |
| --- | --- |
| MEMBERID | EVENTID |

FK FK

**Database Management System (DBMS) Selection**

Members of the group used several DBMS systems throughout the course of this project. Initially it was thought that the group would use the standard DBMS provided with the suggested WAMP/MAMP package. However, as the group members conducted research, some other DBMS options were used and considered. This section will provide details around all DBMS systems used, and the systems(s) ultimately chosen.

1. **MySql Workbench 6.3**

After initially considering using the WAMP/MAMP package, the group decided to explore the possibility of implementing the Entity Framework 6 with MVC. This option was extensively researched and was ultimately rejected based on the fact that not all group members would be able to set up and implement this option. During this process it was suggested to use mysql workbench as the DBMS. Once the decision was made to reject the Entity Framework as an option, **phpMyadmin**, which was the standard DBMS included in the WAMP/MAMP package became the default DBMS.

1. **DbForge**

One of the group members had previous experience with **DbForge**. After experiencing issues when migrating/exporting the database through **phpMyadmin** this member chose to try **DbForge** as an alternative option. Since the others had taken the time to learn **phpMyadmin**, they continued with that DBMS, while the other member chose to use **DbForge** as their default DBMS.

1. **phpMyadmin**

As mentioned above, after some trial and error it was decided by most of the group members to use **phpMyadmin** as the default DBMS.

**SQL Statements**

**CRUD**

For the standard transactions: Create, Retrieve, Update, Delete functions were implemented in PHP. These functions were designed take parameters in order to simulate “generics”. The goal here was to re-use these functions for all of the standard repetitive queries. In all examples of these functions, the table ‘member’ will be used. Some standard Functions and their respective queries can be examined below:

1. The RowCreate function below is a function designed to create a row in a table. The parameters required are:

* $database which is the database
* $table: is the table which the row is to be inserted into
* $value\_array: is an array containing all the attributes for a row in this table

The RowCreate function takes these the parameters and concatenates them into a query string which eventually becomes the query we send to the database.

FUNCTION: function RowCreate ($database, $table, $value\_array)

QUERY: **"INSERT INTO transaction (transactionid, date, memberid, inventorynum, amount, totals, cashierid) VALUES ('45', '2015-12-05', '0', '0', '5', '25', '1234')"**

1. The RowUpdate function below is a function designed to update a row in a table. The parameters required are:

* $database which is the database
* $table: is the table which the row is to be inserted into
* $key\_array: is an array containing any primary keys for the table
* $update\_array: is an array containing all the updated attributes for a row in this table

The RowUpdate function takes these parameters and concatenates them into a query string which eventually becomes the query that is sent to the database.

FUNCTION: function RowUpdate($database, $table, $key\_array, $update\_array)

QUERY: **"UPDATE member SET memberid = '1234', firstname = 'Mike', middlename = ‘S’ '', lastname = ‘Smith’ '', phonenumber = ‘403-123-1234’ '', streetnum = ‘123’'', streetname = ‘my street’ '', city = ‘Calgary’ '', postalcode = ‘B1B-1B1 '', rating =’5.0’ '', renewaldate =’2015-01-14 '', datejoined =’2014-01-01’ '', password = ‘password'', active = '' WHERE memberid = '1234'"**

1. The TableRetrieve function below is a function designed to retrieve a table from the database. The required parameters are:

* $database: is the database
* $table: is the table which the row is to be inserted into

The TableRetrieve function takes these parameters and concatenates them into a query string which eventually becomes the query that is sent to the database.

FUNCTION: function TableRetrieve ($database, $table)

QUERY: **"SELECT \* FROM member”**

1. The getNextID function below is a function designed to retrieve the maximum value in a column. It is used when we want to know the next value of an integer ID. The required parameters are:

* $database: is the database
* $table: is the table which the row is to be inserted into
* $column: is the column to retrieve the maximum value from

The getNextID function takes these parameters and concatenates them into a query string which eventually becomes the query that is sent to the database.

FUNCTION: function getNextID($database, $table, $column)

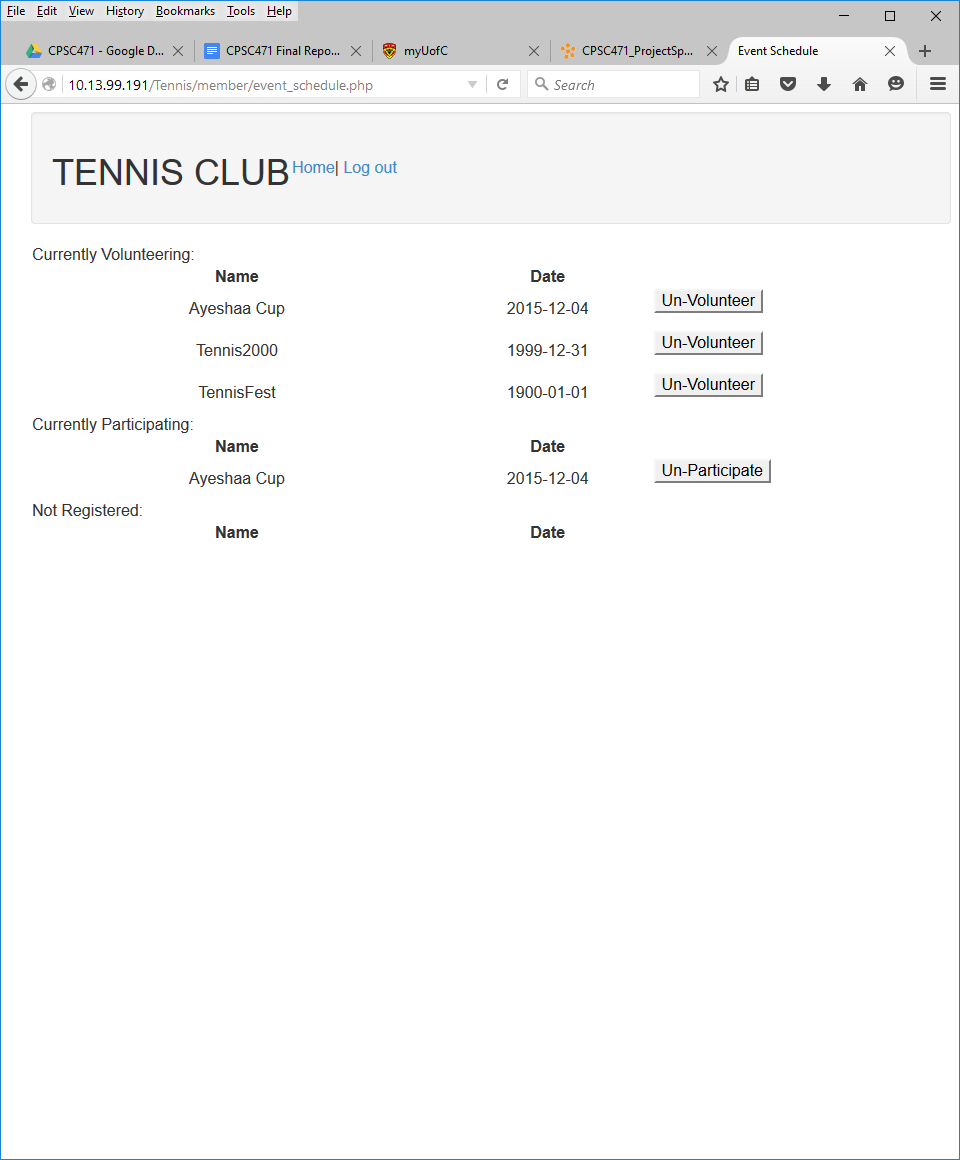
QUERY: **“SELECT max (“memberid”) as ‘memberid’ FROM ‘member’ “**

**Additional SQL**

This section will describe the SQL queries that were necessary in addition to the standard Create,

Retrieve, Update, and Delete (CRUD) queries. In most cases, the CRUD functions are sufficient. In fact, there are only 2 pages where additional queries were required. Below these queries are described along with the pages where they are used. The query strings in all cases are in “PHP” from and include in some cases, variables which would only be available inside of “PHP” code. This format was chosen over the standard SQL query format in order to be able to easily relate the queries to the functions they belong. It is considered a trivial exercise to convert these queries from PHP to SQL

This page lists all events which have taken or will take place at the club. From this page, club members can see what events are coming up as well as what events have taken place. The user can volunteer or participate in any the events. The user can also choose to “un-volunteer”, “un-participate” from events which they have signed up for.



**Query 1:**

Description: This query is used to retrieve and display the events that a user has volunteered for

**"SELECT e.id, e.name, e.date FROM event e WHERE EXISTS (SELECT \* FROM event, volunteers\_for WHERE memberid = '$memberid' and e.id = eventid)";**

**Query 2:**

Description: This query is used to retrieve and display the events that a user has signed up to participate in.

**"SELECT e.id, e.name, e.date FROM event e WHERE EXISTS (SELECT \* FROM event, participates\_in WHERE memberid = '$memberid' and e.id = eventid)";**

**Query 3**

Description: This query is used to retrieve and display the remaining events for which the user has not volunteered for and has not signed up to participate in.

**"SELECT e.id, e.name, e.date FROM event e WHERE NOT EXISTS (SELECT \* FROM event, volunteers\_for v, participates\_in p WHERE ". "(p.memberid = '$memberid' and e.id = p.eventid) or  . "(v.memberid = '$memberid' and e.id = v.eventid))";**

Queries 4-7 (below) are there to facilitate an update to the database as a result of the participate button being clicked.

**Query 4**

Description: This query is used to retrieve the events that a member is participating in.

**"SELECT \* FROM participates\_in WHERE memberid='$memberid' and eventid='$eid'";**

**Query 5**

Description: This query is used to retrieve a row count of events used in query 6

**"SELECT \* FROM event WHERE id='$eid'";**

**Query 6**

Description: This query is used to insert a row into the participates\_in table in the database when a member clicks the “participate” button.

**"INSERT INTO `mydb`.`participates\_in` (`memberid`, `eventid`) VALUES ('$memberid', '$eid')";**

**Query 7**

Description: This query is used to remove a row from the participates\_in table in the database when a memper clicks the “un-participate” button.

**"DELETE FROM participates\_in WHERE memberid='$memberid' and eventid='$eid'";**

Queries 8-11 (below) are there to facilitate an update to the database as a result of the volunteer button being clicked.

**Query 8**

Description: This query is used to retrieve the events that a member is volunteering for.

**"SELECT \* FROM volunteers\_for WHERE memberid='$memberid' and eventid='$eid'";**

**Query 9**

Description: This query is used to retrieve a row count of events used in query 8

**"SELECT \* FROM event WHERE id='$eid'";**

**Query 10**

Description: This query is used to insert a row into the volunteers\_for table in the database when a member clicks the “volunteer” button.

**"INSERT INTO `mydb`.`volunteers\_for` (`memberid`, `eventid`) VALUES ('$memberid', '$eid')";**

**Query 11**

Description: This query is used to remove a row from the volunteers\_for table in the database when a member clicks the “Un-Volunteer” button.

**"DELETE FROM volunteers\_for WHERE memberid='$memberid' and eventid='$eid'";**

Query 12 below is used to extract the firstname and lastname attributes from a concatenated first and last name of a member.

**Query 12**

Description: This query is used to retrieve the memberid attribute from the member table in the database. The values firstname and lastname are extracted from a dropdown list of concatenated first and last names. The dropdown list is populated and found on the transactionTest.php pate

**"SELECT memberid FROM member WHERE firstname = '".$firstName. "' AND lastname = '".$lastName."'";**

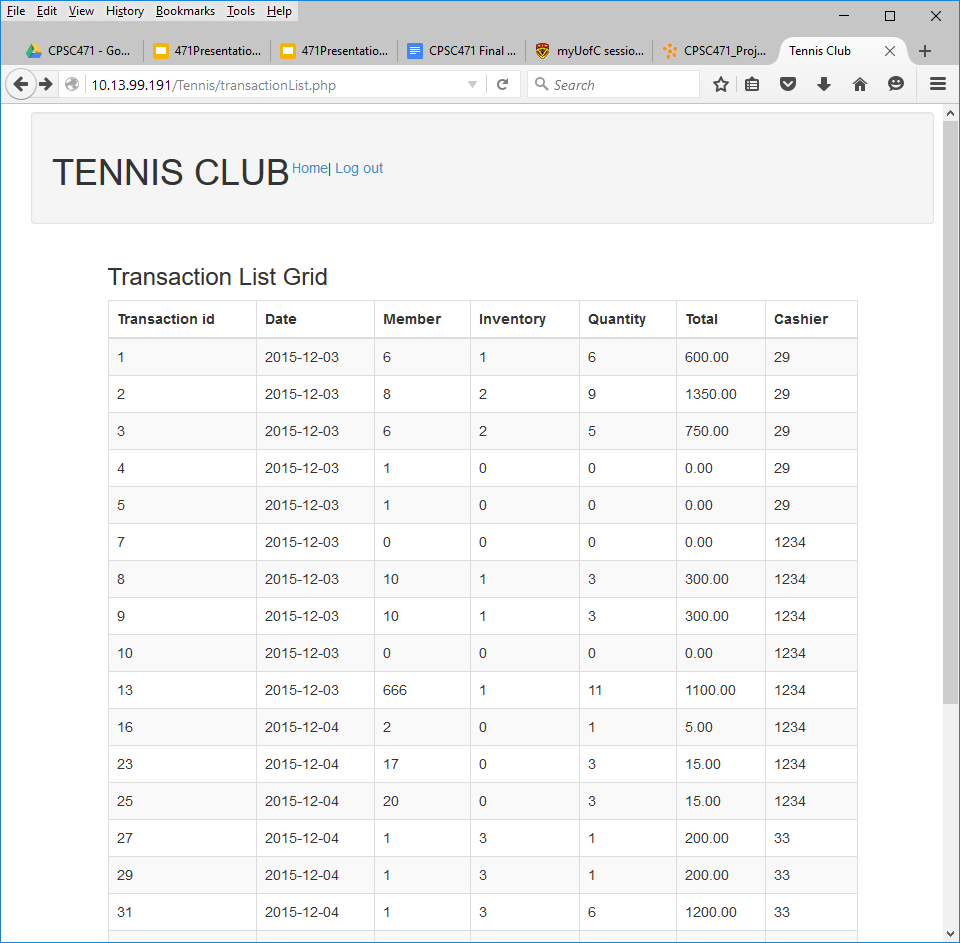
Query 13 below is a join across 4 tables. It is used to “lookup” codes from the transactionList.php page and repost them in a grid on the transactionLookup.php page. An example is transaction id 23 from the transactionList.php page: note that the Member value is 17, the Inventory value is 0 and the Cashier value is 1234. Using this query makes it possible to display these values in a more useful manner on the transactionLookup.php page, where the Name value is Imani (corresponds to Member value 17), The Item Name value is Tennis Ball (corresponds to Inventory value 0), and The Cashier Fname, and Cashier Lname is Mike and Simister respectively (corresponds to Cashier value 1234) and

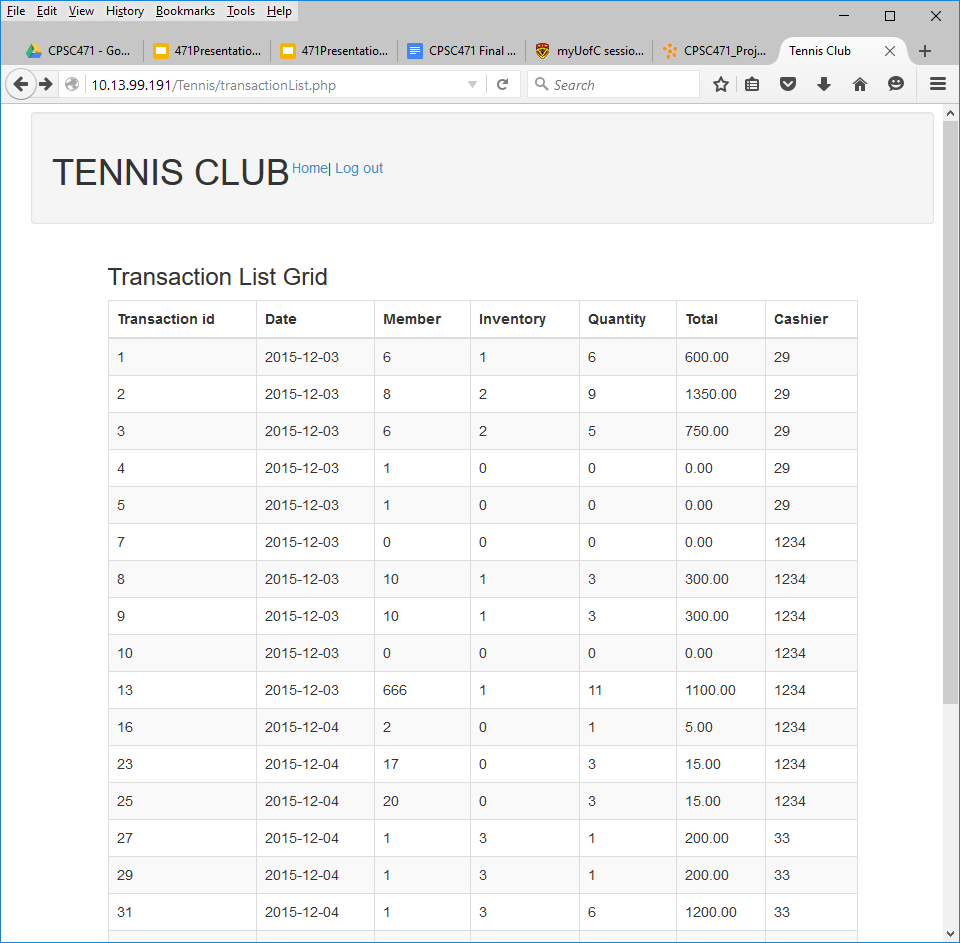
**Query 13**

Description: This query is used to retrieve Text values obtained from a JOIN across 3 tables based on their integer identifiers.

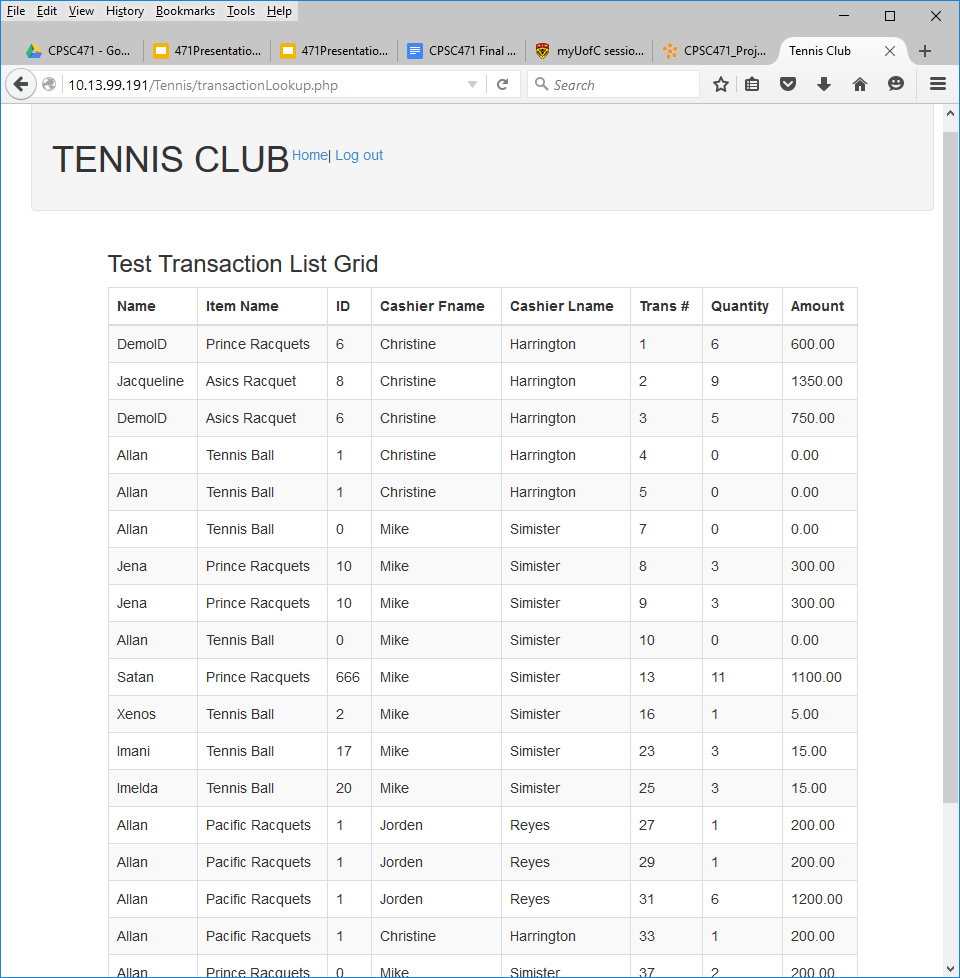
**"SELECT m.firstname AS 'testName', i.name, t.memberid, a.firstname, a.lastname, t.transactionid, t.amount, t.totals FROM member AS m, inventory\_items AS i , transaction AS t, member AS a WHERE m.memberid = t.memberid AND i.inventorynum = t.inventorynum AND a.memberid = t.cashierid ORDER BY t.transactionid;”**

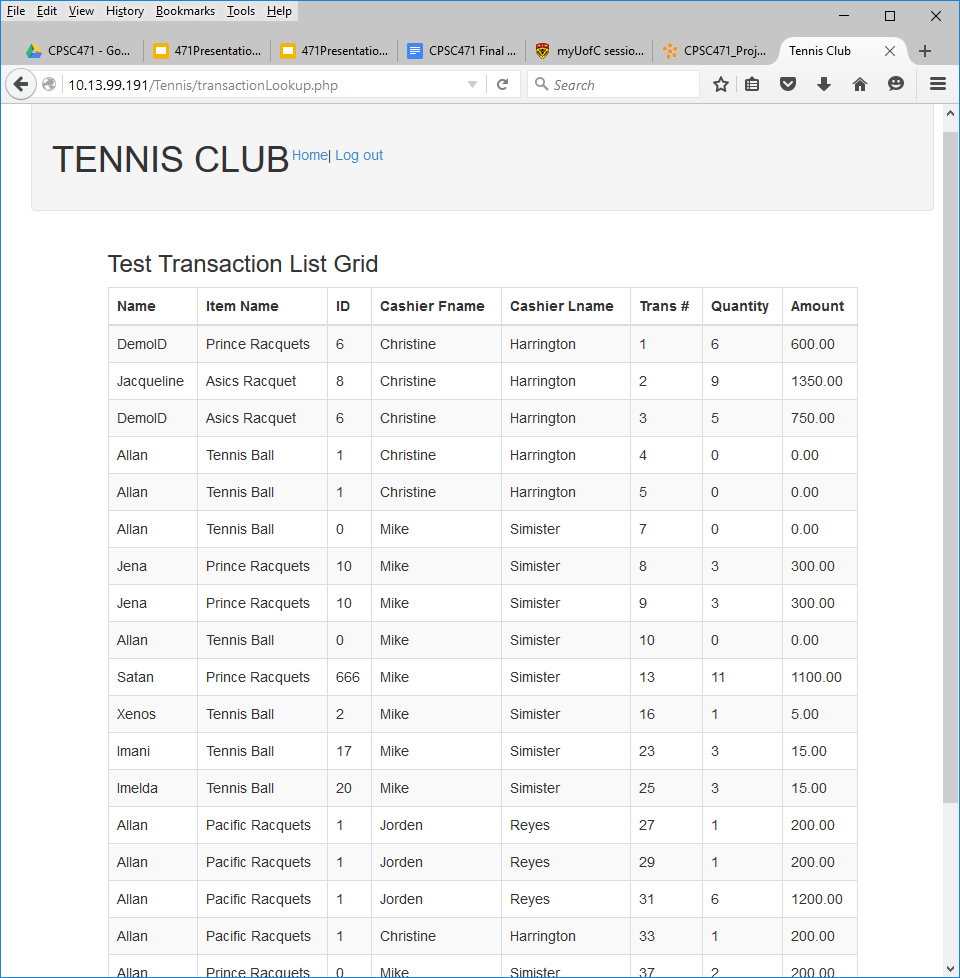
**Screenshot of the transactionList.php row:**





**Screenshot of the transactionLookp.php row:**

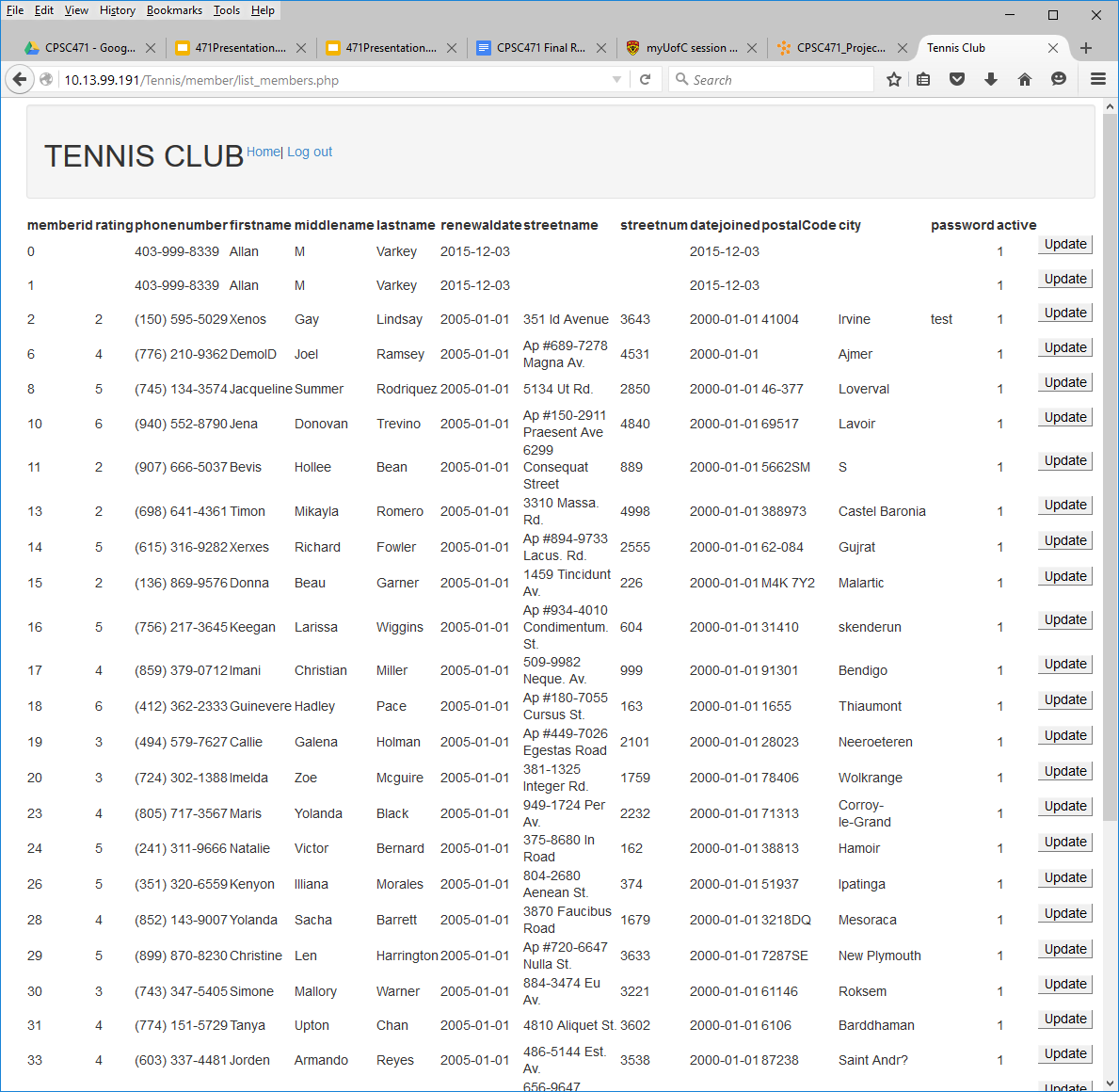




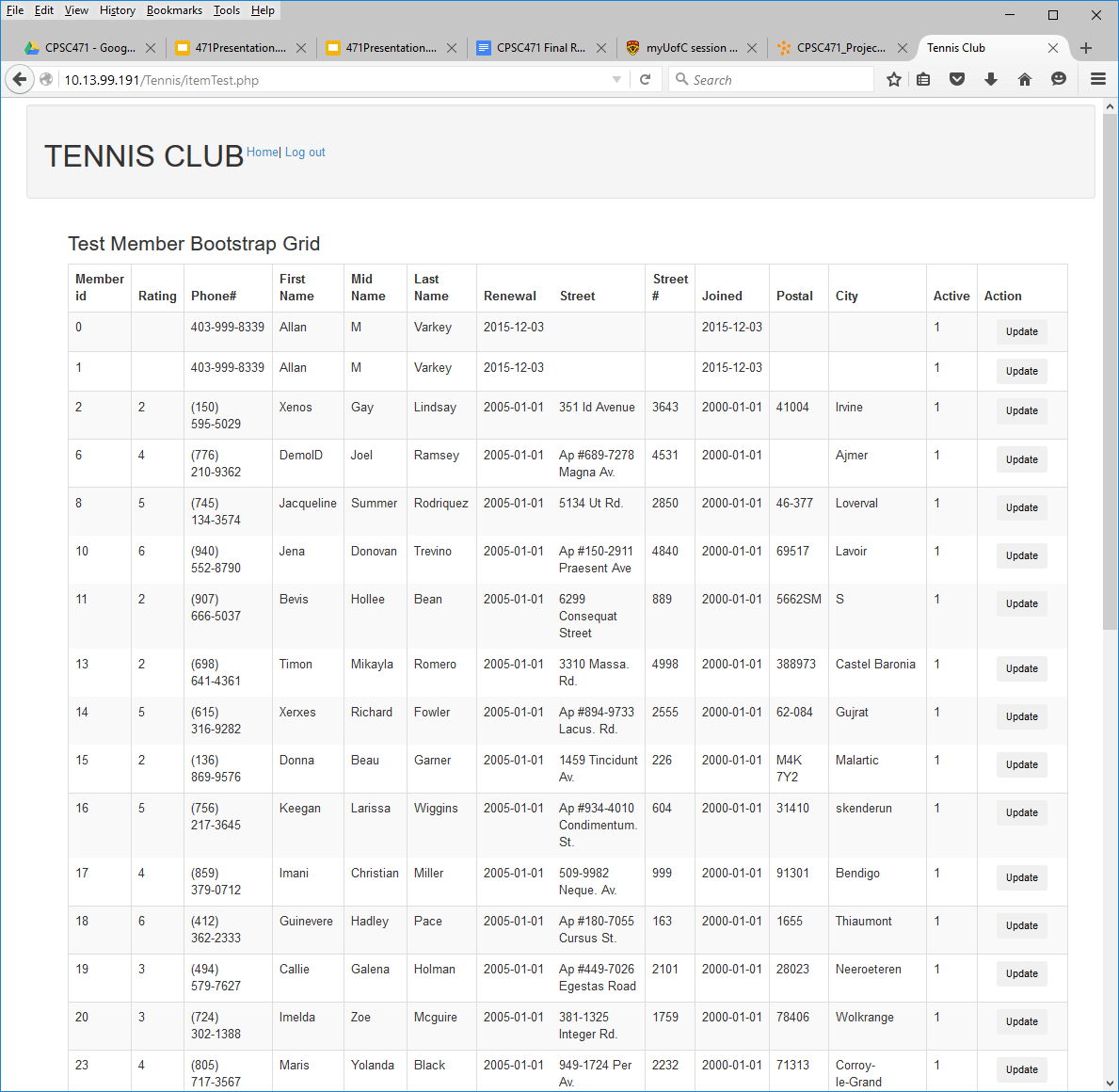
**IV: USER INTERFACE (UI)**

Bootstrap components along with standard HTML tags and .CSS settings were used to create the UI. In some cases information was displayed using standard HTML. This choice was a function of the amount of time available and in some cases the programmers’ experience. It was found that when trying to display several rows and columns of data, that a grid component vastly improved the way the data was displayed. While it was time consuming to learn how to integrate some of the Bootstrap components, in the case of a grid, it was time well spent.

**Members List displayed using standard HTML:**



**Members List displayed using Bootstrap Grid:**



The general look and feel of the UI came as a result of experimenting with the Entity Framework 6. The HTML code generated from this experiment was found to be ideal and was adopted for the project. There was some experimentation among various components. This experimentation could be furthered should the project continue, however it would be up for discussion with the end user as far as how much time and sophistication is required.

**V: CONCLUSION**

The application provides some basic functionality. It has the beginnings of a framework that could be used to make a complete system. Some next steps would include:

* CRUD functionality for all relations
* Lookup tables to display values in common terms instead of their codes
* Reports to provide analysis on the items sold and purchase trends of members as well as simple time period financial/sales information
* Ability for members to modify some of the attributes in the member table
* Display but also restrict the ability to modify particular fields such as memberid, inventory number. (This display/restrict functionality will be required for most tables in the schema)

There is a fair bit to be done if the system were to be formally proposed to an end user. However it the application evolves to that point all group members will have the ability to continue to work on it if it is their choice to do so.