Webbski Rental Management System

Designed by: Andy Rose

CS 420 – Database Management Systems

Spring 2011

Table of Contents

1 Problem Decription 2
1.1 Introduction 2
1.2 Problem Domain 2
1.3 Functionality 3
1.4 Software Development Plan 4
2 Analysis and Conceptual Design 5
2.1 Entities and Relationships 5
2.2 Functional Dependencies 5
3 Logical Design 6
3.1 Relations and Attributes 6
3.2 Functional Dependencies ..... **Error! Bookmark not defined.**
3.3 Decomposition Into Normal Forms 6
3.4 Alternatives ..... **Error! Bookmark not defined.**
4 Implementation, Testing, & Demonstration 7
5 Summary and Conclusions 9
6 References and Appendix 10
6.1 Appendix A – Relation Schema 10

1 Problem Decription

1.1 Introduction

This project is based on the need for a better system to manage the rental process at the Webbski rental shop. Webbski is a privately owned ski school and the Summit At Snoqualmie ski resort. It is actually the biggest, and one of the oldest, of all the private ski schools on the mountain. They are the only private school to have their own rental shop. Having always been a private, family-run company, they do not have an implemented P.O.S. system for their rentals. In fact, they have no computer system at all.

In their system, users pay for a certain number of rentals, based on the number of classes they are scheduled to attend. A custom notecard with their name, information, and number of rentals remaining is created and stored in a box. Each time the customer travels to the mountain for a class, the rental shop workers give them their equipment and write down the numbers of the equipment on the card. The card is then placed in an “out” box, where it will stay until the equipment is returned.

With the rising cost of labor, a computer system will help cut running cost. Having witnessed the system first hand, I can say that there are much easier ways to conduct this operation. This project is meant to help the company keep a more stable inventory of the rental equipment and hasten the process of renting to customers.

1.2 Problem Domain

Currently, the rental shop at Webbski uses custom note cards for each renter in their system. The cards track what equipment the person rents out on what day. They are stored in boxes according to the day the renter is signed up to come in. Old fashioned systems such as this leave opportunities for human inconsistencies and pose some major modern day problems.

The first problem faced is the cards themselves. The renters returning gear have been out in the snow and, along with their equipment, are commonly soaked. Having paper note cards to hold important information around all of this water is a summons for disaster. Also, with the world “going green”, many would say that using all of these paper cards would be a waste. Lastly, the cards can easily be lost and entering wrong data is a frequent occurrence. This is not because of the workers, but the swarms of renters that come in at the same time.

Another issue with the system is keeping track of inventory and renter history. Currently, there is not a nightly check of inventory. The system is to simply check to make sure that no rental cards remained out from being checked in. Being that there is no inventory log, the company would not even know if equipment were to be stolen or lost. Also, sometimes issues come up such as a card got lost or, more often, somebody thinks they purchased more rentals than they have left on their card. In these situations, there is no way to look up the history for the equipment, renter, or transactions.

Lastly, though the rental shop charges for their rentals, these payments aren't tracked in any way. The only information that gets recorded is how much money is made in a given day. The transactions and the sales history cannot be viewed. Having these records is extremely important to a business. It can be useful information to cater better to the customers, and also to build strategies for a better business.

1.3 Functionality

The main users of this program will be those who work in the Webbski rental shop. Being that the system has been in place for so long, it will be important to make sure that it has a similar feel to what the users are familiar with. The key functions the user needs to perform are listed below.

- Rent equipment
- View available equipment
- View equipment checked out
- View all active renters
- Complete a transaction for a renter
- Print and View transaction history
    - All transactions by date period
    - Renter specific transactions
- View specific renter information
- Add/remove equipment
- Change the current season and session
- Add a renter
    - From database add to current season and session
    - New Renter
- Print or view an inventory
- Make equipment inactive for repairs
- Check that Renters have signed a waiver

1.4 Software Development Plan

To create the database for this project, Microsoft Access will be used. This is only to create the Relations and attributes. The database relations table can be seen below in Appendix A. The program itself will have no reliance on Microsoft Access. All of the queries to the database will come from the code. The reason for using the access database is for communication with C# language. The program will be coded in C# and will be set up for a local system. It is best to keep the program local as a desktop application because there is only one computer that it will be used on and it is important to keep the personal information of the Renters secure.

2 Analysis and Conceptual Design

2.1 Entities and Relationships

The main two entities that are dealt with by this program are the renters and the equipment. Both of these will have their own tables within the database, as well as their own personalized ID to be used as a primary key. In those tables, there will be specific attributes, describing all of the necessary information of the entity. These will be described in the next section.

There is no direct relationship between these two entities, but they are both necessary to track the equipment. Looking outside the conventional entity, there is a need for storing each individual rental. This will be used for reference, logging, and inventory. The rental entity is directly related to the renter and the equipment entities, using their primary keys as subkeys and a transaction number as a primary key. Each rental will have a flag telling whether the equipment is out or in.

The last entity that needs to be recognized holds the monetary transactions. This needs to be separate from rentals because users will purchase multiple rentals at once for the season. By separating the two, the program will not waste space on empty values and transaction lookup will be much faster. The only relationship for the transaction entity resides with the renter making the transaction.

2.2 Functional Dependencies

The functional dependencies are very straightforward for this database. The rental entity depends on having a renter, as well as a minimum of one equipment object. The transaction entity depends on having a renter, which may or may not be the person purchasing the transaction. For example, say the renter is 5 years old, obviously their parent will be paying, but the rentals still go to their name. The transaction entity can be completed however without ever having an actual money transaction. The monetary transaction and this P.O.S. system are different, but can later be compared for transaction amount accuracy.

3 Logical Design

3.1 Attributes

The attributes for each entity can be seen below in Appendix A. The number of attributes for the renter is extensive for a system such as this, but the company did not want to lose the personal connection it possesses with its customers. One such attribute that would pertain to this service oriented view is ‘MoneyDue’. The ski school is located up on the hill, about a 15 minute snowy walk from the road. If customers are to forget their wallet, the owner of the company would not like to have them go all the way back to the car to get it. This is for the customer service, as well as to keep things moving and not have instructors waiting for a class member for 30 minutes. The other reason is for tracking damaged equipment. If somebody misused the equipment, they have to pay a replacement cost. Giving customers an I.O.U. tab is a regular occurrence with this private business.

3.2 Decomposition Into Normal Forms

As described in section 2, the transactions are separated from the rentals in the database to avoid empty information and conglomerated functionality. This would cause high amounts of empty space in the database and searching would take much longer. By separating the two, the database is in normal form.

4 Building The System

4.1 Implementation

To keep things simple, the main functionality of the program, pertaining to renting equipment, is all kept on the initial screen. This includes a table of renters, sorted alphabetically by last name, a table of available rental equipment, sorted by the rental number, and a table for equipment checked out, also sorted by rental number. Each table can be organized by any of the listed attributes.

There is a search function for the renters, in which you can type in any attribute to search by, or select a specific attribute to search, which is good for statistics and referencing. As an added functionality to the renter search, selecting the searched user will change the checked-out equipment table to contain only items checked out by that renter. This joins the searching function of the renters and checked-out table, making the program more simple and quicker to use, while also taking up less room on the form.

The search function for the available rental equipment works a little more differently. Because there are only several different types of equipment, typing information, such as the equipment number would take too long. By allowing the user to select different types of equipment using radio buttons for sport and type, scrolling to the equipment number would be much faster than typing.

The last portion of the main screen is button functionality. There exist four buttons on the main page that increase speed and functionality. The first button is to rent the equipment. For this button to function, the user must have a renter and an equipment item selected. The renter must also not have any of that equipment type checked out as well. Also, if the user already has equipment checked out from a previous day that never got checked back in, of any type, the program throws a warning to the user asking them to inquire the renter about the missing equipment.

The next two buttons are part of the search function. One button is simply a search button to locate the query in the database. The other button's functionality is to clear the search. This eliminates navigation button cluttering and provides a fast way to return the screen to its optimal searching speed design. More importantly though, it assures that there will not be any item selected on the renter list when not searching. This is important because the checked-out table directly depends on the renter that is selected.

Other functionalities of the program are contained in the file menu. This includes all of the transactions that a user can make. Amongst other various additions, it also gives the user the ability to add equipment objects or renters. Other key features included in the file menu include settings to select the season, backing up or restoring the database, and viewing information stored in the database.

4.2 Testing

Testing for the program came in several different phases, the first of which was testing the functionality of the user interface. This was just normal use-case testing for the buttons and different features. This testing was first because before implementing the database into the project, the interface first had to be created.

Once the user interface was completed, the next step was to incorporate the database. The testing for this included querying information from different tables to fill the information on the main page as well as the searching functions. Early in this testing phase, the search function also included the search for equipment. Through testing, I found that it was much faster to separate the search for the renters and equipment. By doing this, the speed of the search increased greatly and the user was able to use the program more efficiently. Another feature that spawned from this testing was the ability to search by attribute, making the search faster.

One problem that was faced when testing the searching function was errors and loss of information that came from searching the database using multiple queries on multiple tables simultaneously. To correct this, a method for concatenating a query had to be added to the code. This method gives the ability to join specific information from different tables into a new table, which would then be displayed to the user. This is what made it possible to separate available equipment from that which had been rented out. This method is called twice for every update to the information tables on the interface, once for available equipment, and once for checked-out equipment.

4.3 Demonstration

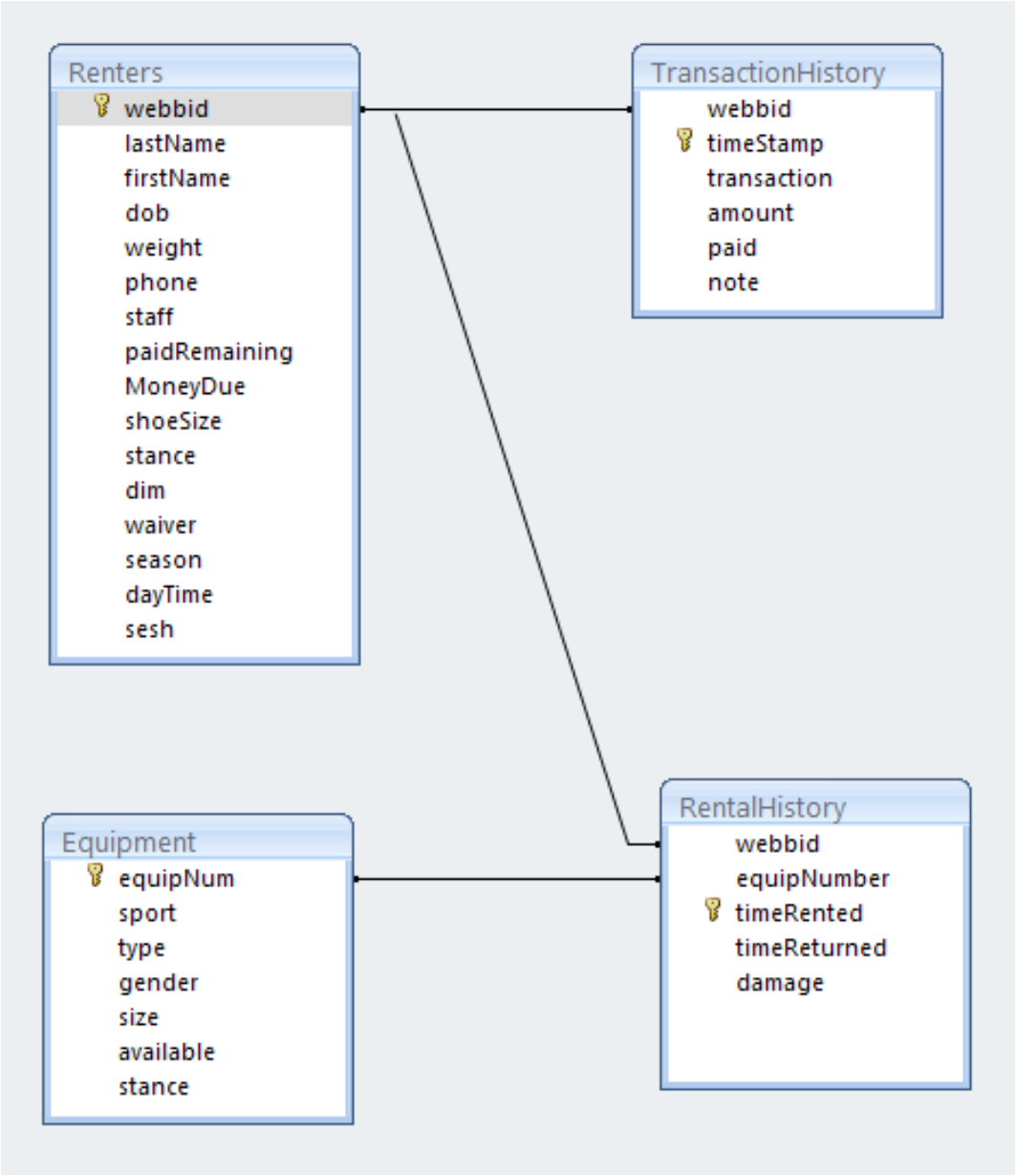
Please see Appendix C and Appendix D for images of the program.

5 Summary and Conclusions

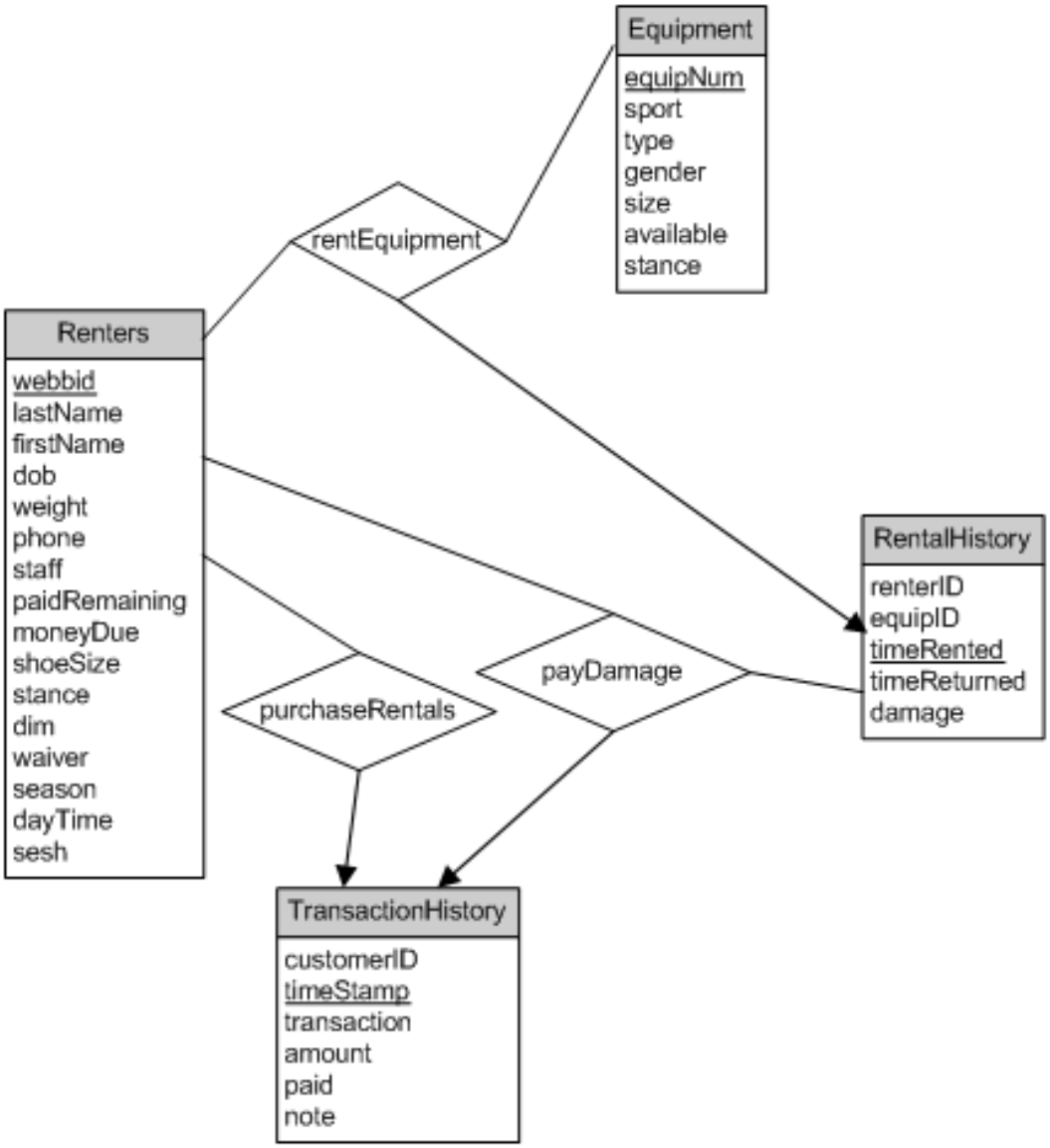
The functionality of the program is all there. This includes the ability to load the information from the database and create queries to search through the information. Everything in the program works to its full potential and has all of the components to be instantiated. However, having less than three months to develop this program, the testing was not as extensive as it should have been for a system such as this. The program has not been tested with a large dataset, which it probably would never see due to the small number of entries in the projected lifespan of being in use at the company. The program has also not been tested for durability, so if it is used beyond the capabilities it was built for there is no telling what errors would possibly occur. All time-constraint related issues aside, there are many ways that this program queries the database and is a good example of what dynamic database access looks like.

6 References and Appendix

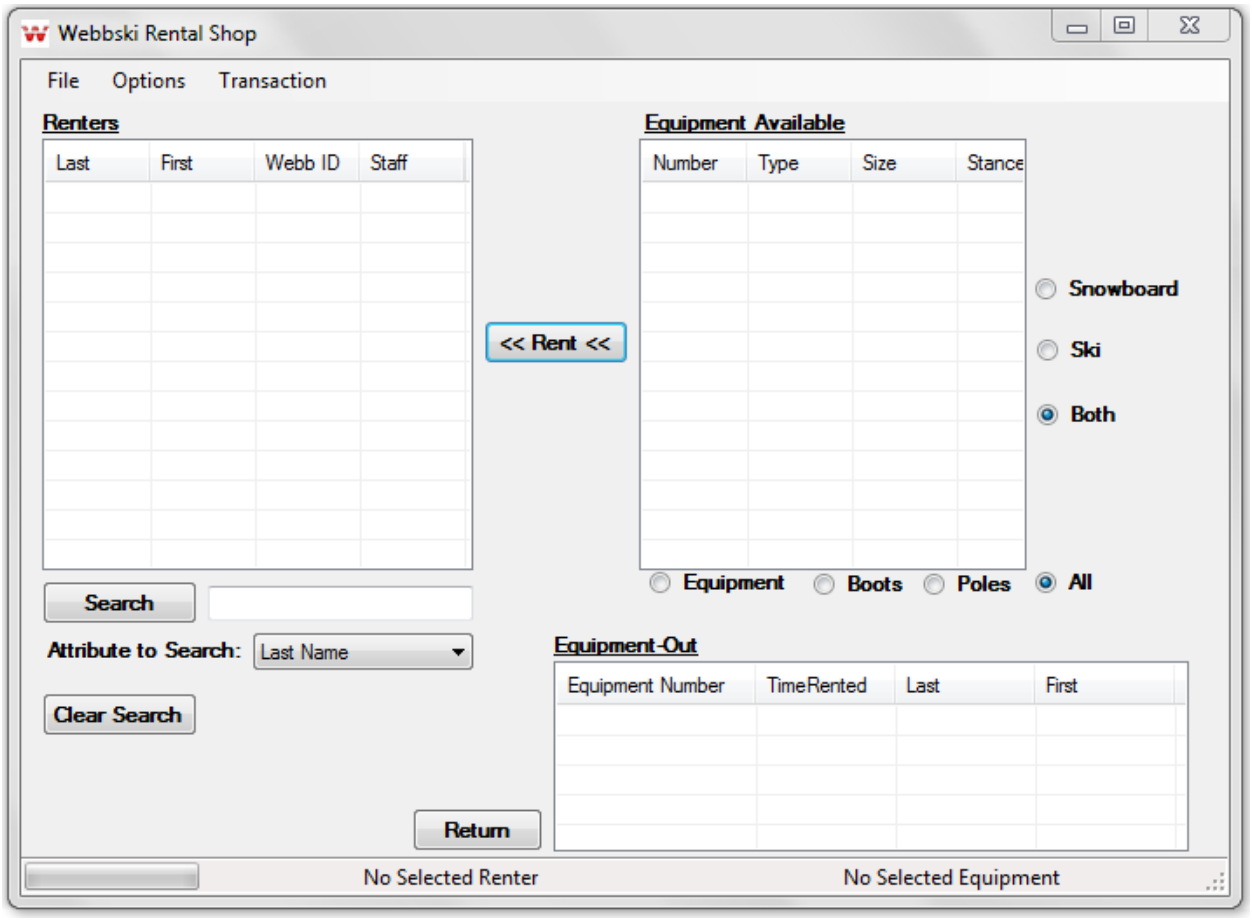
6.1 Appendix A – Relation Schema



6.2 Appendix B – Entity-Relationship Diagram



6.3 Appendix C – Main Screen



6.4 Appendix D – Add Renter Screen