Michael Jones

CS275-400 Final Project Summer 2014

**CS 275 Project**

**Outline**

This is a database for Legos. It is designed so families can list Lego sets that they would like to share and share them. It is very similar to a library. Except there are no due dates and no central storage facility (the library). Its main purpose is to allow users to add sets to the library and check out sets from friends. Users check out Legos by emailing the person who has it and requesting to pick it up.

It is important to track the owner of and the person checking out the Lego set. They might not be the same person. Important note: the owner of a set is allowed to checkout his/her own set. Note: an owner does not have to have a set available for checkout, but a set must have an owner. If a Lego set belongs to me I have ownership of the Lego set. By allowing it on this site I, the owner, need to know at all times where my property is. That is why we check them out and track who currently has the set. The history of who has had the set before is not important as it will be returned to the owner after each person checks it out. Just like checking out a book at the library you have to take it back to the Library (or owner in this case) before it can be checked out again. It is not possible to change the owner, set number, or description of a set once added and the program will only allow one instance of each set. The Legos have a checkout attribute that can change based on the checkout status. Note: a Lego set can be removed from the database but only by its owner.

The site also contains a method of posting photos of a set a user may have built and a missing piece tracker. With the photos page the user may want to post a photo of the completed set, or maybe they took Jabba’s Sail Barge and turned it into a fairy princess house they could display it here. The purpose of the missing piece tracker is to make bulk missing piece orders to save on Lego’s outrageous shipping charges.

The missing piece section allows the users to enter their requested pieces one at a time to the database. It will track the user requesting the piece, element number and description of the piece. If the user finds the piece or no longer needs they may delete the request as well.

**Database Outline**

There are five tables in the database. They are as follows.

LibraryUsers

LegoCollections2

UserPhotos

UserMissing

MissingPiece

LibraryUsers is the central table in the database. Its primary key ‘id’ is used as a reference in nearly all tables. ‘Id’ is also an auto incrementing integer. This table stores the user name, password, and email for each user. Nothing in this table can be changed. It is designed to be permenant.

LegoCollections2 reflects the library’s collection of Legos. Every set in the database is housed here. Its primary key is ‘id’ and is auto incrementing. It stores Lego’s proprietary set number, description of the set, owner of the set and whether or not the set is checked out. If a set is checked out we will also know who is checking it out. LegoCollections2 has two items that have a relationship to another table. LibraryUsers id has a direct relationship with LegoCollections2.owner. The relationship here is simple. LegoCollections2.owner stores owners ‘id’ to represent who owns the item. LegoCollections2.owner is a foreign key reference to LibraryUsers.id.

The other relationship between LibraryUsers and LegoCollections2 is on the checkout attribute. LegoCollection.checkOut references the username attribute of LibraryUsers. It does not have to have a username in it however. CheckOut works like this.

LegoCollections2.checkOut holds either the string ‘available’ or

the string representation of the LibraryUsers.userName.

Due to checkOut holding ‘available’ there is no direct constraint

held on the member.

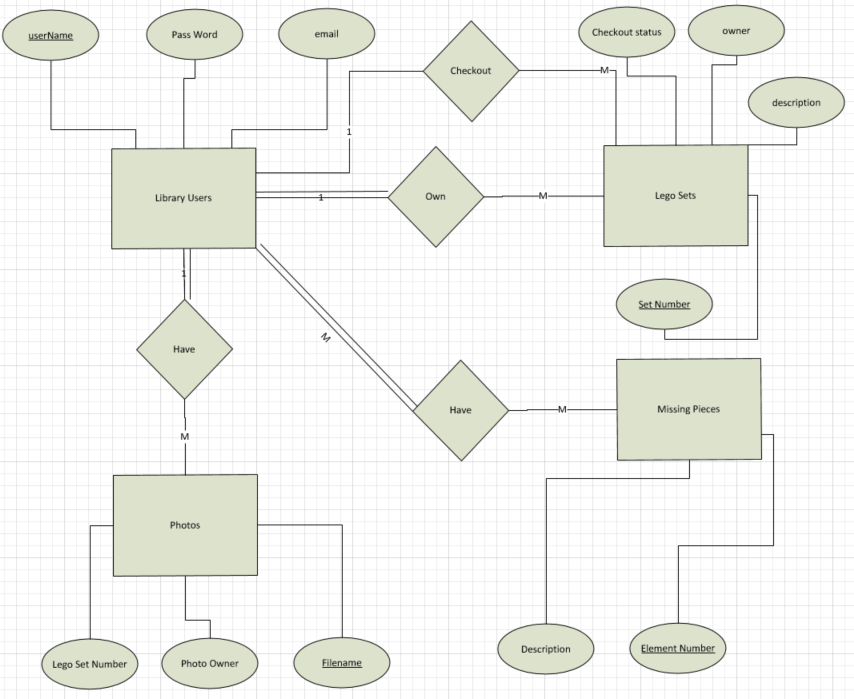
So when a Lego is checked out the php will take the username of the person checking it out and store it in the checkOut field as a string. So if username Michael checks out a set the checkout field for that set will reflect ‘Michael.’ When that set is returned it will reflect ‘available.’ We are not concerned with the checkout history as the set will always be returned to the owner when returned.

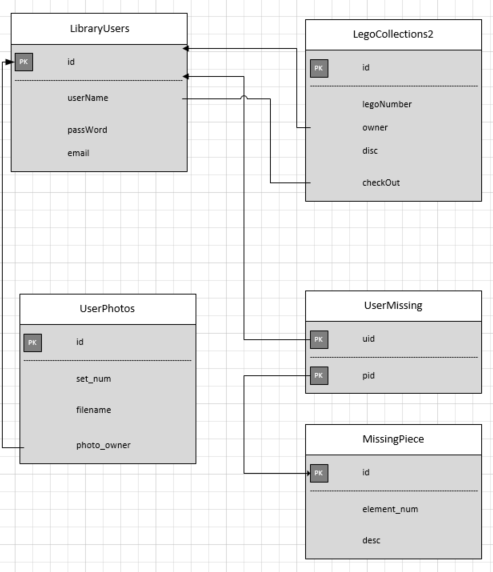
The table MissingPiece has a relationship with LibraryUsers. It contains two fields inputted by the user: element number and part description. Element number is based off of Legos’ unique part numbering system. Note Lego refers to pieces as elements. The part description is pretty straightforward as well. For instance 2x2 plate would indicate a flat plate with a total of four bumps.

The relationship with MissingPieces and LibraryUsers is our only many to many relationship. A user can request many different parts and parts can belong to many different users. In other words I might need element number 4600294 and another user might need it as well. That’s the Lego bone piece by the way. There will be two entries in the table with the same element number, but each entry will have a different ‘id.’ We use the table UserMissing as a bridge between these two tables. UserMissing has two entries a ‘uid’ and a ‘pid.’ Both ’uid’ and ‘pid’ are primary and foreign keys. ‘Uid’ references LibraryUsers.id and ‘pid’ references MissingPiece.id. ‘Pid’ also has the trait of on delete cascade.

The last table is UserPhotos. Here users can upload photos of their creations and everyone can see them. The UserPhoto table has a primary key called ‘id’ which is an auto incrementing integer. It also contains a path to the photo called filename. All photos are kept in a folder called UserPhotos. The filename is the path to that file. The table also holds set\_num which is the set number that the photo is based off of, photo\_owner who is the creator of the photo. Photo\_owner is a foreign key to LibraryUsers.id that on delete cascades.

**ER Diagram**



**Database Schema**

**Table Creation Queries**

CREATE TABLE LibraryUsers (

id int(11) NOT NULL AUTO\_INCREMENT,

‘userName’ varchar(255),

‘passWord’ varchar(255),

‘eMail’ varchar(255),

Primary Key (id)) ENGINE= INNODB;

CREATE TABLE `LegoCollections2` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `legoNumber` int(11),  
 `owner` varchar(255),  
 `disc` varchar(255),  
 `checkOut` varchar(255),  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB

|  |  |
| --- | --- |
|  | CREATE TABLE `UserPhotos` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `filename` varchar(255) NOT NULL,  `set\_num` int(11) NOT NULL,  `photo\_owner` int(11) NOT NULL,  PRIMARY KEY (`id`),   FOREIGN KEY (`photo\_owner`) REFERENCES `LibraryUsers` (`id`) ON DELETE CASCADE ) ENGINE=InnoDB |

CREATE TABLE `UserMissing` (  
 `uid` int(11) NOT NULL,  
 `pid` int(11) NOT NULL,  
 PRIMARY KEY (`uid`,`pid`),  
 FOREIGN KEY (`pid`) REFERENCES `MissingPiece` (`id`) ON DELETE CASCADE,  
 FOREIGN KEY (`uid`) REFERENCES `LibraryUsers` (`id`)  
) ENGINE=InnoDB

CREATE TABLE `MissingPiece` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `quantity` int(11),  
 `element\_num` int(11,  
 `part\_desc` varchar(255),  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB

**General Use Queries**

**User creation [create\_user.php]**

Check to see if username already exists.

SELECT LibraryUsers.userName   
FROM LibraryUsers

WHERE LibraryUsers.userName = [username entered]

Username does not exist store password hash, username and email

INSERT INTO LibraryUsers (userName, passWord, eMail)

VALUES ( [username], [hash], [email])

Select all user info for storage as Session data from the entry created. Select \* used because I use all fields in row.

SELECT \*

FROM LibraryUsers

WHERE userName = [username]

**Login [session.php]**

Check user entered username and password against database and if correct set

session variables.

SELECT LibraryUsers.id, LibraryUsers.userName, LibraryUsers.passWord

FROM LibraryUsers

WHERE userName = [username entered on login]

**Add Photos [processupload.php]**

Insert photo into UserPhotos

INSERT INTO UserPhotos (filename, set\_num, photo\_owner)

VALUES ([newfilename],[set number],[owner])

**Photo Manipulation [DataBase\_Inter.php]**

See all photos of set number user selected.

SELECT UserPhotos.filename

FROM UserPhotos

WHERE UserPhotos.set\_num = [user selected set number]

Show the entire photo table in print out form No actual pictures are displayed

SELECT UserPhotos.set\_num, UserPhotos.filename, LibraryUsers.userName

FROM UserPhotos

INNER JOIN LibraryUsers ON LibraryUsers.id = UserPhotos.photo\_owner

**Add Legos [DataBase\_Inter.php]**

Check to see if a Lego set exists in the database. Only one instance of a set is allowed.

SELECT LibraryUsers.id, LibraryUsers.userName, LibraryUsers.eMail, LegoCollections2.owner, LegoCollections2.legoNumber, LegoCollections2.disc, LegoCollections2.checkOut

FROM LibraryUsers INNER JOIN LegoCollections2

ON LibraryUsers.id = LegoCollections2.owner

WHERE LegoCollections2.legoNumber = [Lego set number]

If it doesn’t exist in the table add it.

INSERT INTO LegoCollections2 (legoNumber, owner, disc, checkOut)

VALUES ( [set number], [owner (the user logged in)], [disc], [available])

**Print Lego Collection table [DataBase\_Inter.php]**

Select all info from LegoCollections2 and userName from LibraryUser.

SELECT LegoCollections2.LegoNumber, LibraryUsers.userName, LegoCollections2.disc, LegoCollections2.checkOut

FROM LibraryUsers INNER JOIN LegoCollections2 ON LibraryUsers.id = LegoCollections2.owner

**Search for a Lego set in the database. [DataBase\_Inter.php]**

Searches for a set by set number in the database.

SELECT LibraryUsers.id, LibraryUsers.userName, LibraryUsers.eMail, LegoCollections2.owner, LegoCollections2.legoNumber, LegoCollections2.disc, LegoCollections2.checkOut

FROM LibraryUsers INNER JOIN LegoCollections2 ON LibraryUsers.id = LegoCollections2.owner

WHERE LegoCollections2.legoNumber = [desired Lego Set]

**Print out a table of users** **[DataBase\_Inter.php]**

Display to screen username and email of all users

SELECT userName, eMail FROM LibraryUsers

**Check out a Lego** **[DataBase\_Inter.php]**

Search for the Lego set desired.

SELECT legoNumber, owner, disc, checkOut

FROM LegoCollections2

WHERE legoNumber = [desired lego set]

If available Update status to “checked out”

UPDATE LegoCollections2 SET checkout = [username of user who wants to check out]

WHERE legoNumber = [lego number to check out]

**Return a Lego** **[DataBase\_Inter.php]**

Search to see if set if checked out.

SELECT legoNumber, owner, disc, checkOut

FROM LegoCollections2

WHERE legoNumber = [Lego set to return]

If Lego set is checked out update status to available

UPDATE LegoCollections2 SET checkout = 'available'

WHERE legoNumber = [checked out status now available]

**Print the specific Lego Set that was modified. Used when updating checked out status [DataBase\_Inter.php]**

Obtain data from specific set.

SELECT LegoCollections2.LegoNumber, LibraryUsers.userName, LegoCollections2.disc, LegoCollections2.checkOut

FROM LibraryUsers

INNER JOIN LegoCollections2 ON LibraryUsers.id = LegoCollections2.owner

WHERE legoNumber = [set number that was updated]

**Delete a Lego Set [DataBase\_Inter.php]**

Search for the Lego set to be deleted. Also getting the username to be sure that user owns the set. Will not allow non-owner to delete a set.

SELECT LibraryUsers.id, LibraryUsers.userName, LibraryUsers.eMail, LegoCollections2.owner, LegoCollections2.legoNumber, LegoCollections2.disc, LegoCollections2.checkOut

FROM LibraryUsers INNER JOIN LegoCollections2

ON LibraryUsers.id = LegoCollections2.owner

WHERE LegoCollections2.legoNumber = [set number to be deleted]

Delete this item

DELETE FROM LegoCollections2

WHERE legoNumber = [set number to be deleted]

**Add a Lego piece to the Missing Piece table. [DataBase\_Inter.php]**

Insert missing piece into missing piece table

INSERT INTO MissingPiece(quantity, element\_num, part\_desc)

VALUES ([quantity], [element number], [part description])

Insert missing piece into UserMissing table and setup foreign key references

INSERT INTO UserMissing(uid, pid)

VALUES ([owner],[last element’s id using insert\_id from insert above])

**View Part Listing [DataBase\_Inter.php]**

Get items from MissingPieces including owner of the request

SELECT MissingPiece.quantity, MissingPiece.element\_num, MissingPiece.part\_desc, LibraryUsers.userName

FROM LibraryUsers

INNER JOIN UserMissing ON LibraryUsers.id = UserMissing.uid

INNER JOIN MissingPiece ON MissingPiece.id = UserMissing.pid"

**Delete a part from the MissingPiece table [DataBase\_Inter.php]**

Search for the missing piece based on owner and element number

SELECT MissingPiece.id

FROM LibraryUsers

INNER JOIN UserMissing ON LibraryUsers.id = UserMissing.uid

INNER JOIN MissingPiece ON MissingPiece.id = UserMissing.pid

WHERE MissingPiece.element\_num = [element number selected] AND LibraryUsers.id = [user logged in]

Delete the item if you own it and only delete one instance. php code prevents this.

DELETE FROM MissingPiece

WHERE MissingPiece.element\_num = [element number to be deleted]

**Show all the parts that you have requested [DataBase\_Inter.php]**

Search for Missing Pieces based off of logged in owner.

SELECT MissingPiece.quantity, MissingPiece.element\_num, MissingPiece.part\_desc

FROM LibraryUsers

INNER JOIN UserMissing ON LibraryUsers.id = UserMissing.uid

INNER JOIN MissingPiece ON MissingPiece.id = UserMissing.pid

WHERE LibraryUsers.userName = [logged in userName]

**See who ordered the same part. (demo of many to many) [DataBase\_Inter.php]**

Search for all parts with same element number

SELECT MissingPiece.quantity, MissingPiece.element\_num, MissingPiece.part\_desc, LibraryUsers.userName

FROM LibraryUsers

INNER JOIN UserMissing ON LibraryUsers.id = UserMissing.uid

INNER JOIN MissingPiece ON MissingPiece.id = UserMissing.pid

WHERE MissingPiece.element\_num = [element number searched for]