College of Saint Benedict & Saint John’s University

Computer Science Department

GABeS

Phase 3

Team Potatoes

Grant Boyer, Kyle Olson, Tom Husen

[This page has been intentionally left blank]

Table of Contents

Database Design ……………………………………………………………………………………………………………………... 4

EER Diagram ………………………………………………………………………………………………………………... 4

Relational Map Diagram ...…………………………………………………………………………………………... 5

Physical Database Design ………………………………………………………………………………………………………… 6

Issues Faced During Phase .……………………………………………………………………………………………………… ?

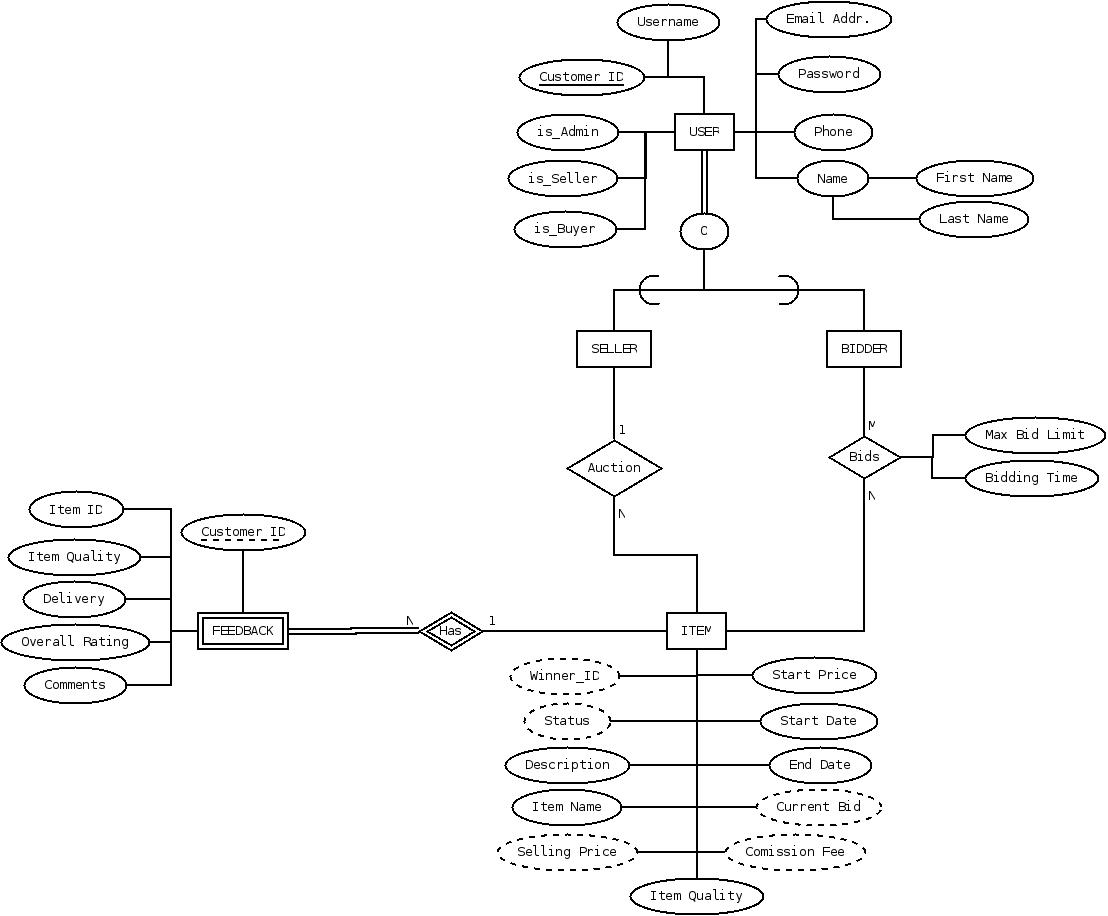
Task Decomposition ………………………………………………………………………………………………………………… ?

Meeting Minutes …………..………………………………………………………………………………………………………… ?

November 8, 2016 ..……………………………………………………………………………………………………... ?

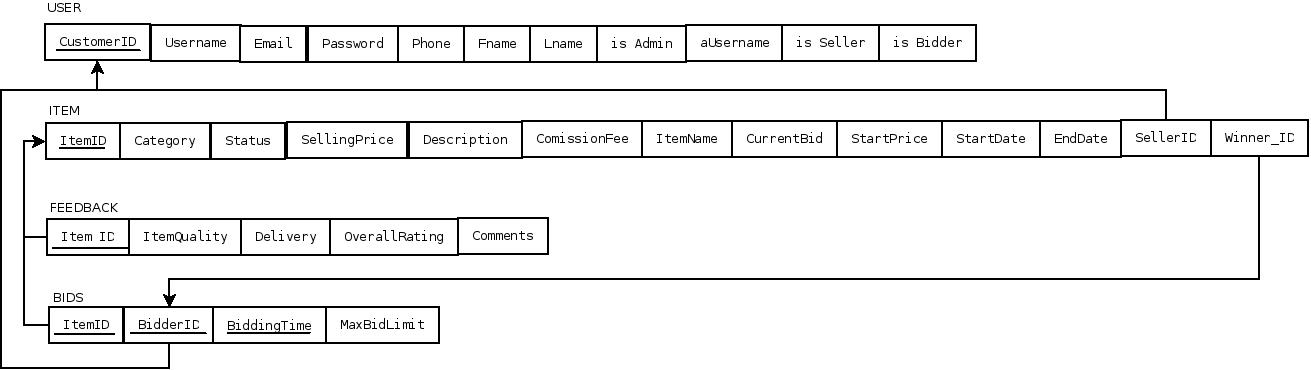
November 13, 2016 ……………………………………………………………………………………………………... ?

November 14, 2016 ……………………………………………………………………………………………………... ?

Database Design *-* EER Diagram

Explanation of EER Diagram

Above is an updated version of our EER Diagram. Since our original submission of this diagram, we have restructured how *Feedback* links to the database, deleted various unnecessary entities, and where possible changed to derived attributes. We also changed how the *Admin* will work. Rather than having a separate entity for the Admin, we decided to set a variable *is\_Admin* within the *User* entity. This allows us to make use of the *User* functionalities much more easily and not over complicate the database.

Logical Database Design – Relational Mapping

Explanation of Relational Map

Above is our up-to-date Relational Map. Since our original map design, we have made several changes. Including eliminating a few unnecessary items, and more properly mapping the relations involving foreign keys. We reduced from 7 entities to 4 so this new database design if far more efficient, simple, and all around a better solution.

Physical Database Design

Due to the lengthy nature of this phase, a bulk of the code, comments, tests, etc. are located in 3 file that will accompany this submission. All of the information included in these files (except test cases) are located in our team Oracle connection.

**POTATOES\_Create\_and\_Populate\_Tables.sql** when ran will create all of the tables and populate them with relevant sample data that can be used throughout including tests

**POTATOES\_Components.sql** contains all of the triggers, functions, procedures, views, and sequences that are used throughout the system.

**POTATOES\_TestCases.sql** contains test cases for all of the functionalities in the system. Please note: if you try to run the entire script it will not always work. Best practice is to run each test individually.

All of the creation tables for the database are in the first accompanied file, and they include domain constraints, integrity constraints, primary keys, foreign keys, etc.

System Functionality

*For this section, each relevant webpage that we generated as a part of phase 1 of this project, will be described in terms of how it will relate to our SQL database, as well as provide any necessary SQL queries that will help complete the desired functionality.*

*Please Note: Nearly every single one of these functionalities has an associated View/Procedure/Function/Trigger building upon these simple SQL queries. These expanded components are located in the next section of this report.*

Admin Login

Description – This is the page that is reached from the homepage when an admin selects to login. They will enter a *username* and *password*. Our query will then check the database to see if there is an admin user corresponding to the provided credentials. If successful, it will output all information about that user. If unsuccessful, it will output an empty table

SQL:

Select \* From GABES\_USER Where USERNAME='tehusen' and PASS='p@$$word' and IS\_ADMIN=1;

User Login

Description – This is the page that is reached from the homepage when a user selects to login. They will enter a *username* and *password*. Our query will then check the database to see if there is a non-admin user corresponding to the provided credentials. If successful, it will output all information about that user. If unsuccessful, it will output an empty table

SQL:

Select \* From GABES\_USER Where USERNAME='tehusen' and PASS='p@$$word' and IS\_ADMIN=0;

Admin Commission Report

Description – This page will allow administrators to get all relevant information about a user, calculate their average overall rating, and their total commissions.

SQL:

SELECT x.USER\_ID, u.USERNAME, u.FIRST\_N, u.LAST\_N, u.EMAIL, AVG(f.OVERALL\_RATING) AS Seller\_Rating, SUM(x.STATUS) AS Commissions

FROM GABES\_USER u, GABES\_ITEM x

FULL OUTER JOIN GABES\_FEEDBACK f

ON f.ITEM\_ID = x.ITEM\_ID

WHERE u.USER\_ID = x.USER\_ID

GROUP BY x.USER\_ID, u.USERNAME, u.FIRST\_N, u.LAST\_N, u.EMAIL

ORDER BY x.USER\_ID;

Sales Summary Report

Description – This page will allow administrators to get all records for any item that has been sold (has a STATUS = 1)

SQL:

SELECT ITEM\_CATEGORY, ITEM\_ID, ITEM\_NAME, SELLING\_PRICE, COMMISSION\_FEE

FROM GABES\_ITEM

WHERE STATUS = 1;

Manage Users

Description – This page has 2 functionalities. The first is to display all users in the database for the admins to see. They can then link to other functionalities such as edit users, etc. The second functionality is to add a new user to the database. It uses an insert statement to specify username, email, etc. and accesses our *Sequence* for creating new users, to auto-generate the next USER\_ID number in that sequence.

SQL:

-This is the First part of the functionality

CREATE OR REPLACE VIEW GABES\_VIEWUSERS AS

Select u.USER\_ID, u.USERNAME, u.FIRST\_N, u.LAST\_N, u.EMAIL

From GABES\_USER u

ORDER BY u.USER\_ID;

- This is the 2nd part of the functionality

INSERT INTO GABES\_USER VALUES (new\_user\_seq.NEXTVAL, new\_username, new\_email, new\_pass,

new\_phone, new\_first, new\_last,new\_admin, new\_admin\_u, new\_sell, new\_buy);

Update Profile

Description – This page is for updating any and all information for a user that is already in the database. For the sake of brevity, we are only including the update statement for updating the *first name* field. Every other update you could do for a user is identical, you just change the field.

SQL:

UPDATE GABES\_USER

SET GABES\_USER.FIRST\_N = p\_firstName

WHERE p\_firstName IS NOT null AND USER\_ID = p\_userID;

Item List

Description – This page is for getting all items that a specific user has listed on the site (in the database). It selects specified information from the ITEM table where the SELLER ID is equal to the user ID

SQL:

SELECT ITEM\_ID, ITEM\_NAME, START\_DATE, END\_DATE, START\_PRICE, CURRENT\_BID, STATUS,

USER\_ID

FROM GABES\_ITEM

WHERE SELLER\_ID = USER\_ID;

Bidder List

Description – This page is for getting all users who have bid on a specific item. It queries the database based on the item ID, and returns all relevant information for people who have bid on a that item

SQL:

Select u.USERNAME as Username, b.ITEM\_ID as Item\_ID, b.MAX\_BID as Max\_Bid,

b.BIDDING\_TIME as Bid\_Time

From GABES\_BIDS b, GABES\_USER u, GABES\_ITEM i

Where b.ITEM\_ID = i.ITEM\_ID and u.USER\_ID = b.BIDDER\_ID;

Update Item Info

Description – This page is for updating the information for a specific item that already exists in the database. Similarly, to *Update User Info*, for the sake of brevity in this report we are only including one update statement (in this case update category).

SQL:

UPDATE GABES\_ITEM

SET GABES\_ITEM.ITEM\_CATEGORY = p\_newCategory

WHERE p\_newCategory IS NOT null AND p\_itemId IS NOT null AND p\_itemId = ITEM\_ID AND

USER\_ID = p\_userID;

Post New Item

Description – This page is for adding a brand new item to the database. It passes through the information that is required for an item in the database, and then using the *Sequence* that we created for the items, it is automatically assigned the next number in the sequence for its item\_id

SQL:

INSERT INTO GABES\_ITEM (ITEM\_ID, ITEM\_CATEGORY, STATUS, SELLING\_PRICE, DESCRIPTION,

COMMISSION\_FEE, ITEM\_NAME, CURRENT\_BID, START\_PRICE, START\_DATE, END\_DATE,

USER\_ID, WINNER\_ID) VALUES (NEW\_ITEM\_SEQ.nextVal, p\_itemCategory , p\_status,

p\_sellingPrice, p\_description, p\_comissionFee, p\_itemName , p\_currentBid,

p\_startPrice, p\_startDate, p\_endDate, p\_userID, p\_winnerID );

Item Search

Description – This is another example of a page where there are several different SQL statements based on various conditions. For the sake of brevity, we will include only one example of these statements. Essentially what is happening in these statements is we are sending through information we want to search by in the *where* section, and then intersecting several different selection statements to get the search results

SQL:

Select \*

From GABES\_SEARCH

Where upper(CATEGORY) LIKE '%BOOKS%'

INTERSECT

Select \*

From GABES\_SEARCH

Where CURRENT\_BID <= 18 OR (BEGIN\_PRICE <= 18.00 and CURRENT\_BID IS NULL);

Bid on Item

Description – At the core of this functionality is adding a new entry into the bids table in the database. We pass through all required fields and a new record is created in the bids table. This is our first example of the CURRENT\_TIMETAMP keyword, which marks the current time on the computer when that statement is run. This will keep track of when they submitted the bid.

SQL:

INSERT INTO GABES\_BIDS VALUES (new\_itemID, new\_bidderID, CURRENT\_TIMESTAMP, new\_maxLimit);

Bid Status

Description – This will get all of the status (still up for auction or not) for items that a certain user has bid on at any point in time.

SQL:

SELECT i.STATUS

FROM GABES\_BIDS b, GABES\_USER x, GABES\_ITEM i

WHERE i.ITEM\_ID = b.ITEM\_ID AND b.BIDDER\_ID = x.USER\_ID

Items Bought

Description – This will get all of the items bought for a specific user. It accomplishes this by getting all of the items who have a winner ID equals to the user ID of that user.

SQL:

SELECT \*

FROM GABES\_ITEM

WHERE WINNER\_ID = USER\_ID;

Leave Feedback

Description – This will add a new entry into the feedback table in the database. We pass through all required information and a new record is created for this piece of feedback.

SQL:

INSERT INTO GABES\_FEEDBACK VALUES(item\_id, quality, deliver, overall, comments);

View Users Ratings

Description – This allows a user to view all of the feedback that they have been given in the system. It queries the feedback table and gets records matching their username

SQL:

SELECT DISTINCT z.USERNAME, y.ITEM\_ID, y.OVERALL\_RATING, y.ITEM\_QUALITY, y.DELIVERY,

y.COMMENTS, z.User\_ID

FROM GABES\_FEEDBACK y, GABES\_USER z JOIN

GABES\_ITEM x

ON z.USER\_ID = x.USER\_ID

WHERE y.ITEM\_ID = x.ITEM\_ID;

SQL Code Components

Please Note: We have created far more than the required 12 SQL components. When planning out this phase, we made a team decision that these views, procedures, etc. would make the database easier to understand and easier for use in the future. For the sake of this report we will include only 12 of our 29 SQL components.

Triggers:

* **GABES\_AUCTION\_ENDED**
* This first trigger executes one of our stored procedures, before inserting into the BIDS table of the database. The process is whenever there is an attempted insertion it will run the stored procedure. In short, it checks if the auction has ended and then executes a series of statements to set various other attributes in the database – effectively ending the auction.
* This works alongside the **Bid On Item** functionality.

CREATE OR REPLACE TRIGGER GABES\_AUCTION\_ENDED

BEFORE INSERT ON GABES\_BIDS

FOR EACH ROW

BEGIN

GABES\_CHECK\_TIME;

END;

* **GABE\_BID\_TRIGGER**
* This second trigger monitors the BIDS table as well. This time after a successful insertion into the table it will modify the ITEM table so that the current bid is up to date.
* This works alongside the **Bid On Item** functionality.

CREATE OR REPLACE TRIGGER GABES\_BID\_TRIGGER

AFTER INSERT ON GABES\_BIDS

FOR EACH ROW

BEGIN

UPDATE GABES\_ITEM

SET CURRENT\_BID = :NEW.MAX\_BID

WHERE GABES\_ITEM.ITEM\_ID =:NEW.ITEM\_ID;

End;

Stored Functions:

* **GABES\_ADMINLOGIN**
* This function takes 2 parameters (username and password) and then queries the database to see if this combination is a valid one, as well as checks to make sure the user queried is an admin. It then returns a 1 upon success, or a 0 upon failure.
* This works alongside the **Admin Login** functionality.

CREATE OR REPLACE FUNCTION GABES\_AdminLogin(param\_user String, param\_pass String) RETURN int AS

temp int := 0;

BEGIN

Select COUNT(\*) into temp

From GABES\_USER

Where USERNAME = param\_user and PASS = param\_pass and IS\_ADMIN = 1;

Return temp;

END;

* **GABES\_USERLOGIN**
* This function takes 2 parameters (username and password) and then queries the database to see if this combination is a valid one, as well as checks to make sure the user queried is not an admin. It then returns a 1 upon success, or a 0 upon failure.
* This works alongside the **User Login** functionality.

CREATE OR REPLACE FUNCTION GABES\_UserLogin(param\_user String, param\_pass String) RETURN int AS

temp int := 0;

BEGIN

Select COUNT(\*) into temp

From GABES\_USER

Where USERNAME = param\_user and PASS = param\_pass and IS\_ADMIN = 0;

Return temp;

END;

Stored Procedures:

* **GABES\_ADD\_USER**
* This procedure takes username, email, password, phone number, first name, last name, admin who created user, and, is\_admin, is\_seller, is\_buyer values as well. It then generates the next user ID value using our *Sequence* that we made (which will be discussed later). It then inserts into the GABES\_USER table the new values.
* This works alongside the **Manage Users** functionality.

CREATE OR REPLACE PROCEDURE GABES\_ADD\_USER(new\_username VARCHAR, new\_email VARCHAR, new\_pass VARCHAR, new\_phone VARCHAR, new\_first VARCHAR, new\_last VARCHAR, new\_admin CHAR, new\_admin\_u VARCHAR, new\_sell CHAR, new\_buy CHAR) AS

BEGIN

INSERT INTO GABES\_USER VALUES (new\_user\_seq.NEXTVAL, new\_username, new\_email, new\_pass,

new\_phone, new\_first, new\_last, new\_admin, new\_admin\_u, new\_sell, new\_buy);

END;

* **GABES\_LEAVE\_FEEDBACK**
* This procedure takes item id, rating for quality, rating for delivery, and overall rating. It also takes any comments about the transaction. It takes these values and then inserts the result into the GABES\_FEEDBACK table.
* This works alongside the **Leave Feedback** functionality.

CREATE OR REPLACE PROCEDURE GABES\_LEAVE\_FEEDBACK(item\_id CHAR, quality CHAR, deliver CHAR, overall

CHAR, comments VARCHAR) AS

BEGIN

INSERT INTO GABES\_FEEDBACK VALUES(item\_id, quality, deliver, overall, comments);

END;

* **UPDATE\_USER\_PROFILE\_EMAIL**
* This procedure takes user id and new email and then updates the corresponding records in the database with the new values.
* This works alongside the **Update Profile** functionality.

CREATE OR REPLACE PROCEDURE UPDATE\_USER\_PROFILE\_EMAIL(p\_userId IN int,p\_email IN String)

IS

BEGIN

UPDATE GABES\_USER

SET GABES\_USER.EMAIL = p\_email WHERE p\_email IS NOT null AND USER\_ID = p\_userID;

END;

* **GABES\_CHECK\_TIME**
* This procedure takes no parameters, and checks all of the items in the ITEM table and if the END\_DATE (end of auction) has passed, then it sets the appropriate values to their new values. Status=1 because auction is over, selling price=current bid because that’s the final bid, and commission fee = (.05 \* current bid) because that’s the commission fee for the system
* This works alongside the **Bid on Item** functionality.

CREATE OR REPLACE PROCEDURE GABES\_CHECK\_TIME AS

BEGIN

Update GABES\_ITEM

Set STATUS = 1, SELLING\_PRICE = CURRENT\_BID, COMMISSION\_FEE = (.05\*CURRENT\_BID)

Where CURRENT\_TIMESTAMP > END\_DATE;

END;

Views:

* **GABES\_BIDDERLIST**
* This view generates a temp table for the username, user id, max bid, and bidding time for all items that a user has posted, and have been bid on.
* This works alongside the **Get Item Info** functionality.

CREATE OR REPLACE VIEW GABES\_BIDDERLIST AS

Select u.USERNAME as Username, b.ITEM\_ID as Item\_ID, b.MAX\_BID as Max\_Bid, b.BIDDING\_TIME as Bid\_Time

From GABES\_BIDS b, GABES\_USER u, GABES\_ITEM i

Where b.ITEM\_ID = i.ITEM\_ID and u.USER\_ID = b.BIDDER\_ID;

* **GABES\_VIEWUSERS**
* This view generates a table with only relevant information for all of the users in the database. It omits unnecessary flags, password, etc.
* This works alongside the **Manage Users** functionality.

CREATE OR REPLACE VIEW GABES\_VIEWUSERS AS

Select u.USER\_ID, u.USERNAME, u.FIRST\_N, u.LAST\_N, u.EMAIL

From GABES\_USER u

ORDER BY u.USER\_ID;

* **ITEM\_LIST**
* This view generates a table with all items in the database. When you call this view you specify the user\_ID and then the table will be only items that that specific user has posted for auction
* This works alongside the **Get Items for User** functionality.

CREATE OR REPLACE VIEW ITEM\_LIST AS

SELECT ITEM\_ID, ITEM\_NAME, START\_DATE, END\_DATE, START\_PRICE, CURRENT\_BID, STATUS, USER\_ID

FROM GABES\_ITEM;

* **SALE\_SUMMARY\_REPORT**
* This view generates a table that has all of the sales information for the database. It is accessed by the admins so they can see which users have sold what, as well as the total sales statistics for the site. Gets a select amount of information from the ITEM table where the status=1 which means the auction has ended
* This works alongside the **Sales Summary Report** functionality.

CREATE OR REPLACE VIEW Sale\_Summary\_Report AS

SELECT ITEM\_CATEGORY, ITEM\_ID, ITEM\_NAME, SELLING\_PRICE, COMMISSION\_FEE

FROM GABES\_ITEM

WHERE STATUS = 1;

Sequences:

Though not required – we found that the sequence component of Oracle SQL was going to be very helpful in our implementation. We have 2 sequences that are nearly identical. Here is the sequence for ITEM\_ID

* **NEW\_ITEM\_SEQ**
* This sequence starts at 1, and each time the .NEXTVAL operation is called on it, it will return the next value in that sequence – until the maximum of 9999999999999 is reached.
* This works alongside all **ITEM** related functions.

CREATE SEQUENCE NEW\_ITEM\_SEQ

MINVALUE 1

MAXVALUE 9999999999

INCREMENT BY 1

START WITH 1

NOCACHE

NOORDER

NOCYCLE

NOPARTITION ;

Issues Faced

During this phase we faced many different issues stemming from SQL Developer not working properly to figuring out how on earth everything would be connected and come together. Ultimately, we were able to vanquish many of these issues and by not cutting corners and working our hardest to ensure our solutions were both efficient and forward thinking, we will be saving ourselves lots of work in the next phase as well as making our product more complete.

When we began working on this phase, we started by going through every website that we designed during phase 1. For each webpage we discussed what exactly the purpose of that page was, and how we though to best accomplish that in terms of SQL implementation. Then we divided all of the tasks so each team member could work on their tasks before bringing them all together into final products. For some of these, issues arose when trying to implement our initial ideas so we had to bounce ideas off each other to determine what would actually be the best approach to the problems. This collaboration about our ‘separate’ parts of the project was immensely helpful as they often applied to more than one task which caused us to be consistent across the board when it came to the implementation.

Another problem we faced was a couple times when testing a SQL query on our team server, we experienced lag and sometimes crashing of our Horizon Client that was running on our personal computer (though not always the Client’s fault). While there is nothing we could really do at the time of crashing, we agreed that if we were each to have a copy of the databases on each of our individual Oracle DB connections, we could test our queries/functions/procedures/etc. without these bugs happening.

When working on the Bid on Item functionality, we were planning on using a Scheduler to execute a stored procedure every 5 seconds or so because we didn’t know how else to perform these functions ‘live’. When we tried creating the scheduler we ran into permissions issues and the Oracle server wouldn’t allow us to create or run the scheduler we needed. The code that we would have used is listed in Appendix A.

Task Decomposition

Grant:

* Responsible for (Components):
  + Admin Commission Report (View)
  + Sale Summary Report (View)
  + Bid Status (View)
  + User’s Rating(s) (View)

Kyle:

* Responsible for (Components):
  + Update Profile (Procedure)
  + Item List (View)
  + Update Item Info (Procedure)
  + Post Item (Procedure) – Additionally Sequence for ITEM\_ID
* Responsible for maintaining Task Spreadsheet

Tom:

* Responsible for (Components):
  + Admin Login (Function)
  + User Login (Function)
  + Manage Users (View)
  + Add New User (Procedure) – Additionally Sequence for USER\_ID
  + Bidder List (View)
  + Leave Feedback (Procedure)
  + Search
* Compiled information into final report

All Team Members:

* Responsible for (Components):
  + Bid on Item
* Creating test cases and documentation to demonstrate the full functionality of the components they created

Team Potatoes Minutes

November 8, 2016

Meeting began at 3:45 am.

**In Attendance**:

* Grant Boyer
* Kyle Olson
* Thomas Husen

**All**:

* Discussed all of the elements that goes into this phase and the best approach for completing them
* Began creating database tables and populating them with sample data values
* Discussed our relational map and EER diagram as they relate to the database and began making appropriate changes

Meeting adjourned at 5:30 am.

Meeting Minutes

Team Potatoes Minutes

November 13, 2016

Meeting began at 4:30 pm.

**In Attendance**:

* Grant Boyer
* Kyle Olson
* Thomas Husen

**Tom**:

* Update EER and Relational Map based on our most recent decisions regarding our database

**All**:

* Review all webpages created during Phase 1 – generate spreadsheet breaking down each page to the tasks it needs to accomplish, how we think to best accomplish that task, comments about the task, and who is going to be responsible for completing that task – Appendix B
* Discuss any changes that are required to be made on the EER diagram or the Relational map

Meeting adjourned at 11:30 pm.

Team Potatoes Minutes

November 14, 2016

Meeting began at 11:15 am.

**In Attendance**:

* Grant Boyer
* Kyle Olson
* Thomas Husen

**Kyle**:

* Manage our task spreadsheet – ensuring all data is up to date with the latest decisions we have made as a group

**Tom**:

* Compile all information into phase report

**All**:

* Continue working on our individually assigned SQL components
* Continue developing test cases to ensure proper functioning of all SQL components

Meeting adjourned at 11:45 pm.

Appendix A:

Begin  
grant create any job to TEAM1;  
grant execute on DBMS\_SCHEDULER to TEAM1;  
grant manage scheduler to TEAM1;  
END;  
  
begin  
    dbms\_scheduler.create\_job(job\_name        => 'CHECK\_AUCTIONS',  
                              job\_type        => 'STORED\_PROCEDURE',  
                              job\_action      => 'GABES\_CHECK\_TIME',  
                              start\_date      => systimestamp,  
                              end\_date        => null,  
                              repeat\_interval => 'FREQ=MINUTELY; byhour=0; byminute=0; bysecond=5;',  
                              enabled         => true,  
                              auto\_drop       => false,  
                              comments        => 'Runs every 5 seconds to run update to check and see if current time is past the end time for any auctions');  
end;

Appendix B: [Link to Full Spreadsheet](https://docs.google.com/spreadsheets/d/1z6nhSXivd2X7Wp-q0HPXdWTKZkxs3ngMJjpG4JW9taw/edit?usp=sharing)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | HTML Page Name | Fuctionality Name | Person | ✔ | Component? (What kind) | Comments |
| (ADMIN) 1 | Homepage | - |  |  |  | directs to user or admin page |
| 2 | Admin Login | Login | Tom | ✔ | Function | Check to see if username and password are in the database, then checks admin flag and if those are true, return true and then the html will redirect to admin homepage |
| 3 | Admin Menu | - |  |  |  | directs to user management page or view reports page |
| 4 | View Reports | - |  |  |  | directs to sales summary report or commission report page |
| 5 | Admin Comission Report | Create Comission Report | Grant |  | View | select all the columns from the user table and display them as a view |
| 6 | Sale Summary Report | Create Sales Report | Grant | ✔ | View | select all the columns from the item table and display them as a view -- check for the sold flag, only display those items |
| 7 | Manage Users | Display current users, add new user | Tom | ✔ | View, Procedure | show all current users, insert new user |
| (USER) 8 | User Login | Login | Tom | ✔ | Function | Check to see if username and password are in the database, then checks admin flag and if that is false, return true and then the html will redirect to admin homepage |
| 9 | User Management Page | - |  |  | - | directs to update profile, selling management, bidding management, leave feedback, view feedback page |
| 10 | Update Profile | update | Kyle | ✔ | Procedure | Update current Users information (MULTIPLE PROCEDURES FOR EACH FIELD, PASS IN THE DATA USING JSP POST METHOD, IN JAVA CREATE PREPARED STATEMENT AND PARSE THE JSP INFO INTO THE APPROPRIATE QUERY FIELDS |
| 11 | Selling Management Page | - |  |  |  | redirects to List Items, add items, and user menu |
| 12 | Item List | Get all items for this user | Kyle | ✔ | View (select from view where...)) | Gets all items for this user, where user id= current user |
| 13 | Bidder List | get specific item info | Tom | ✔ | View | Get all items from bids table where item id = id from itemlist |
| 14 | Item Info | update item | Kyle | ✔ | Procedure | Update the item name, category and description for items already up for auction |
| 15 | Post Item | create item | Kyle | ✔ | Procedure | Run insert statement to add item id (should just be one more than the current item), name.... |
| 16 | Bidding Management | - |  |  |  | redirects to search item, check status, back to user menu |
| 17 | Item Search | ?????? |  |  |  |  |
| 18 | Search Results | table of info from the item search | Team |  | View(From item seach) | Take params from item search page and make view from them |
| 19 | Bid on Item | ??????(CURRENT\_TIMESTAMP gives current time) | Team |  | Procedure, Trigger(s) | Insert New bid into bids table, have triggers for the highest bid, ... set to look for insert into the bids table |
| 20 | Bid Status | Get all items a user bid on | Grant | ✔ | View | Get all items from bids where bidder id = user id, join with item to get other attribute info |
| 21 | List of Items Bought | Get all items won | Kyle | ✔ | View | Get all items from the item table where winner id = bidder id |
| 22 | Leave Feedback | rate item a user won | Tom | ✔ | Procedure | Insert a new rating for a item a user bought, fill out fields in the form and derive the item name |
| 23 | User's Rating | View this users ratings | Grant | ✔ | View | Select all info from feedback table where item.itemid = feedback.itemid and user.id = item.sellerid (JOIN) |