College of Saint Benedict & Saint John's University
Computer Science Department

GABeS

Phase 3

Team Potatoes

Grant Boyer, Kyle Olson, Tom Husen

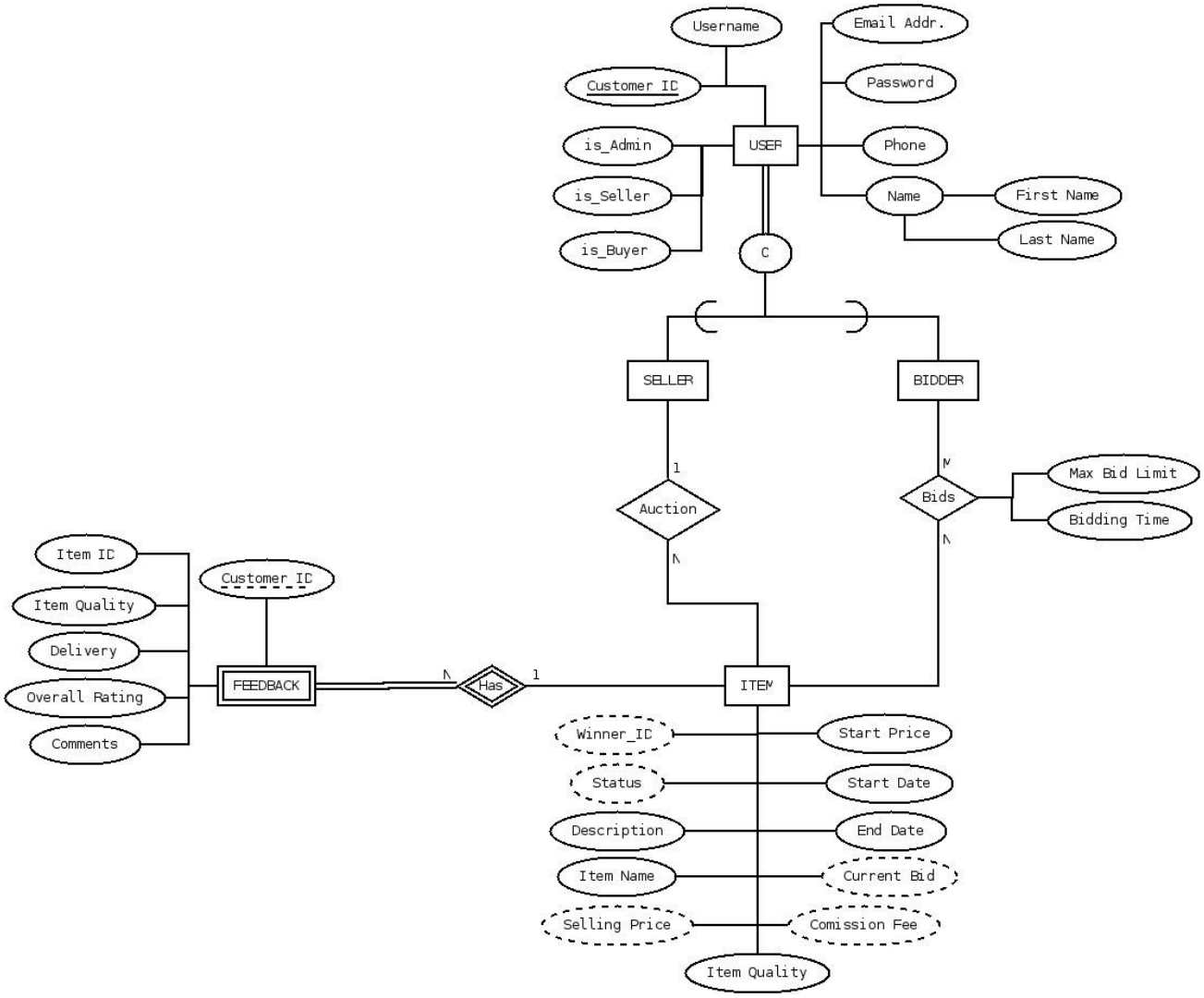
[This page has been intentionally left blank]

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 ..... ?

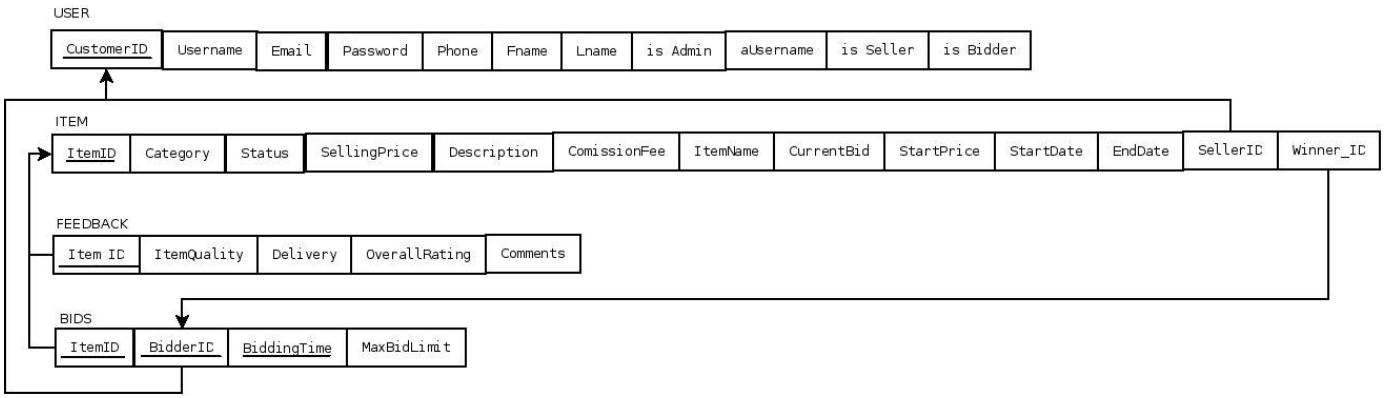
Database Design - EER Diagram



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.

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]

	HTML Page Name	Fuctionality Name	Person	✓	Component? (What kind)
(ADMIN) 1	Homepage	-			
2	Admin Login	Login	Tom	✓	Function
3	Admin Menu	-			
4	View Reports	-			
5	Admin Comission Report	Create Comission Report	Grant		View
6	Sale Summary Report	Create Sales Report	Grant	✓	View
7	Manage Users	Display current users, add new user	Tom	✓	View, Procedure
(USER) 8	User Login	Login	Tom	✓	Function
9	User Management Page	-			-
10	Update Profile	update	Kyle	✓	Procedure
11	Selling Management Page	-			
12	Item List	Get all items for this user	Kyle	✓	View (select from view where...)
13	Bidder List	get specific item info	Tom	✓	View
14	Item Info	update item	Kyle	✓	Procedure
15	Post Item	create item	Kyle	✓	Procedure
16	Bidding Management	-			
17	Item Search	??????			
18	Search Results	table of info from the item search	Team		View(From item seach)
19	Bid on Item	??????(CURRENT_TIMESTAMP gives current time)	Team		Procedure, Trigger(s)
20	Bid Status	Get all items a user bid on	Grant	✓	View
21	List of Items Bought	Get all items won	Kyle	✓	View
22	Leave Feedback	rate item a user won	Tom	✓	Procedure
23	User's Rating	View this users ratings	Grant	✓	View