COMP0022 Project Report: Group 5 Submission

1. Introduction

Within this project, we have designed and built an auction system, “AllAboutCars”, primarily using PHP and MySQL. This report will explain the background behind our database design, a description of our database system, an analysis of the database schema and an explanation for the MySQL queries that we have used.

2. Video

The video that we have created to demonstrate our auction can be accessed via the following URL: **https://youtu.be/MLR5FE5eV4o**

3. Creating Tables and Test Data

The file CreateTablesAndTestData.php can be used to create the MySQL tables used within this project and populate them with test data. Firstly, an empty database “auction” needs to be created in PHPMyAdmin. The file can then be called by typing the URL into the browser. The following MySQL queries are used in order to do this.

```
$seller = "CREATE TABLE IF NOT EXISTS seller (
        `email` varchar(30) NOT NULL,
        `fName` varchar(15) NOT NULL,
        `lName` varchar(15) NOT NULL,
        `password` varchar(40) NOT NULL,
        `phoneNo` varchar(15) DEFAULT NULL,
        `street` varchar(20) DEFAULT NULL,
        `city` varchar(20) DEFAULT NULL,
        `postcode` varchar(10) DEFAULT NULL,
        PRIMARY KEY (`email`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “seller” table if it does not already exist in the ‘auction’ database. It contains personal information about the seller, using “email” as the primary key.

```
$buyer = "CREATE TABLE IF NOT EXISTS buyer (
        `email` varchar(30) NOT NULL,
        `fName` varchar(15) NOT NULL,
        `lName` varchar(15) NOT NULL,
        `password` varchar(40) NOT NULL,
        `phoneNo` varchar(15) DEFAULT NULL,
        `street` varchar(20) DEFAULT NULL,
        `city` varchar(20) DEFAULT NULL,
        `postcode` varchar(10) DEFAULT NULL,
        PRIMARY KEY (`email`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “buyer” table if it does not already exist in the “auction” database. It contains personal information about the buyer, using “email” as the primary key.

```
$category = "CREATE TABLE IF NOT EXISTS category (
        `categoryID` int(2) NOT NULL AUTO_INCREMENT,
        `categoryName` varchar(65) NOT NULL,
        PRIMARY KEY (`categoryID`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “category” table if it does not already exist in the “auction” database. It contains all categories for auctioned items, with “categoryID” as the primary key.

```
    //Items
    $items = "CREATE TABLE IF NOT EXISTS items (
        `itemID` int(11) NOT NULL AUTO_INCREMENT,
        `itemName` varchar(50) NOT NULL,
        `sellerEmail` varchar(30) NOT NULL,
        `image1` longblob NOT NULL,
        `image2` longblob NOT NULL,
        `image3` longblob NOT NULL,
        `description` text NOT NULL,
        `categoryID` int(5) NOT NULL,
        `conditionOfUse` varchar(100) NOT NULL,
        `colour` VARCHAR(30) DEFAULT NULL,
        `gearbox` VARCHAR(30) NOT NULL,
        `fuelType` VARCHAR(30) NOT NULL,
        `initialReg` INT(10) NOT NULL,
        `mileage` INT(10) NOT NULL,
        `doors` INT(3) DEFAULT NULL,
        `seats` INT(3) DEFAULT NULL,
        `acceleration0to60mph` VARCHAR(5) DEFAULT NULL,
        `topSpeedMph` INT(4) DEFAULT NULL,
        `enginePowerBhp` INT(5) DEFAULT NULL,
        `startPrice` decimal(8,2) NOT NULL,
        `reservePrice` decimal(8,2) DEFAULT NULL,
        `startDate` datetime NOT NULL,
        `endDate` datetime NOT NULL,
        PRIMARY KEY (`itemID`),
        KEY `fk_categoryID_items` (`categoryID`),
        KEY `fk_sellerEmail_items` (`sellerEmail`),
        CONSTRAINT `fk_categoryID_items` FOREIGN KEY (`categoryID`) REFERENCES
 `category` (`categoryID`),
        CONSTRAINT `fk_sellerEmail_items` FOREIGN KEY (`sellerEmail`) REFERENC
ES `seller` (`email`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “items” table if it does not already exist in the “auction” database. It contains all relevant item features of a listing, with “itemID” as the primary key. This table has two foreign key constraints: “sellerEmail”, derived from the primary key “email” of the “seller” table, and “categoryID”, derived from the primary key “categoryID” of the category table.

```
$bids = "CREATE TABLE IF NOT EXISTS bids (
        `bidID` int(11) NOT NULL AUTO_INCREMENT,
        `buyerEmail` varchar(30) NOT NULL,
        `itemID` int(11) NOT NULL,
        `bidPrice` decimal(8,2) NOT NULL,
        `bidDate` datetime NOT NULL,
        PRIMARY KEY (`bidID`) USING BTREE,
        KEY `fk_buyerEmail_bids` (`buyerEmail`),
        KEY `fk_itemID_bids` (`itemID`),
        CONSTRAINT `fk_buyerEmail_bids` FOREIGN KEY (`buyerEmail`) REFERENCES
`buyer` (`email`),
        CONSTRAINT `fk_itemID_bids` FOREIGN KEY (`itemID`) REFERENCES `items`
(`itemID`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “bids” table if it does not already exist in the “auction” database. It contains all the bids a buyer makes on an auction, with “bidID” as the primary key. This table has two foreign key constraints: “itemID”, derived from the primary key “itemID” of the items table and “buyerEmail”, derived from the primary key “email” of the buyer table.

```
$listings_watched = "CREATE TABLE IF NOT EXISTS listings_watched (
        `watchID` int(11) NOT NULL AUTO_INCREMENT,
        `buyerEmail` varchar(30) NOT NULL,
        `itemID` int(11) NOT NULL,
        PRIMARY KEY (`watchID`),
        KEY `fk_buyerEmail_listings_watched` (`buyerEmail`),
        KEY `fk_itemID_listings_watched` (`itemID`),
        CONSTRAINT `fk_buyerEmail_listings_watched` FOREIGN KEY (`buyerEmail`)
 REFERENCES `buyer` (`email`),
        CONSTRAINT `fk_itemID_listings_watched` FOREIGN KEY (`itemID`) REFEREN
CES `items` (`itemID`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “listings_watched” table if it does not already exist in the “auction” database. It contains all listings watched by interested buyers, with “watchID” as the primary key. This table has two foreign key constraints: “itemID”, derived from the primary key “itemID” of the items table and “buyerEmail”, derived from the primary key “email” of the buyer table.

```
 $purchase = "CREATE TABLE IF NOT EXISTS `purchase` (
        `purchaseID` int(11) NOT NULL AUTO_INCREMENT,
        `itemID` int(11) NOT NULL,
        PRIMARY KEY (`purchaseID`),
        KEY `fk_itemID_purchase` (`itemID`),
        CONSTRAINT `fk_itemID_purchase` FOREIGN KEY (`itemID`) REFERENCES `ite
ms` (`itemID`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;";
```

This query creates the “purchase” table if it does not already exist in the “auction” database. It contains all successful purchases of items by buyers, with “purchaseID” as the primary key. This table has two foreign key constraints: “itemID”, derived from the primary key “itemID” of the items table and “buyerEmail”, derived from the primary key “email” of the buyer table.

```
$test_data_seller = "INSERT INTO seller (
        `email`, `fName`, `lName`, `password`,
        `phoneNo`, `street`, `city`, `postcode`)
        VALUES (
            'boris.johnson@uk.com',
            'Boris',
            'Johnson',
            SHA('PASSWORD123'),
            '0123456789',
            '10 Downing St',
            'London',
            'SW1A 2AB');";
```

This query inserts some test data into the newly created “seller” table. We must do this to insert test items that will fill the browse web page. The password is encrypted with the SHA() function.

```
$test_data_categories = "INSERT INTO category (`categoryName`) VALUES ('SUV'),
 ('Coupe');";
```

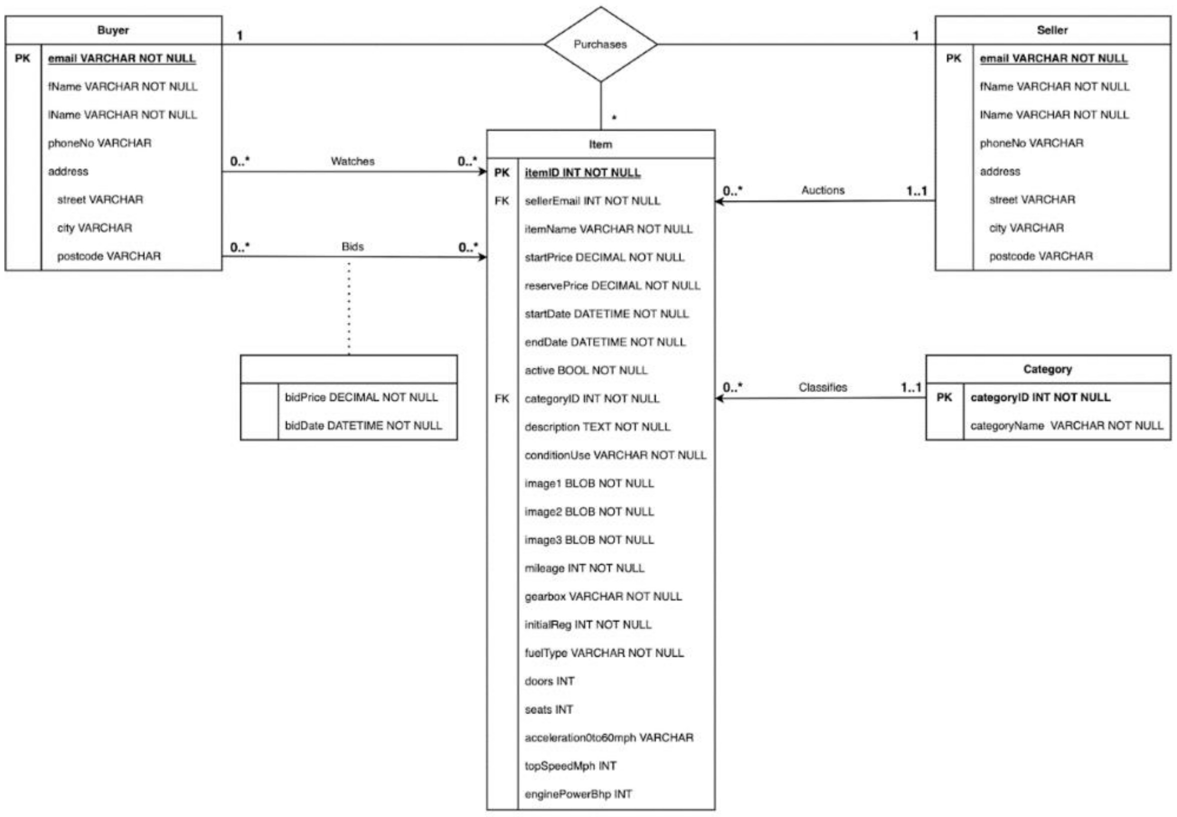
This query inserts some test data into the newly created “category” table. We must do this to insert test items that will fill the browse web page.

```
$test_data_item = "INSERT INTO items (
        `itemName`, `sellerEmail`, `image1`, `image2`, `image3`,
        `description`, `categoryID`, `conditionOfUse`, `colour`,
        `gearbox`, `fuelType`, `initialReg`, `doors`, `seats`,
        `mileage`, `acceleration0to60mph`, `topSpeedMph`,
        `enginePowerBhp`, `startPrice`, `reservePrice`,
        `startDate`, `endDate`)
        VALUES
        ('Audi RS5',
        'boris.johnson@uk.com',
        '$audirs5_img1',
        '$audirs5_img2',
        '$audirs5_img3',
        'Finest German engineering!!',
        2, 'New', 'Green', 'Manual',
        'Electric', 2019, 4, 4, 0,
        '0.8s', 320, 1010, 145000,
        180000, now(), '2021-01-29 14:59:00'),
        ('Mercedes G-Class',
        'boris.johnson@uk.com',
        '$mercedes_img1',
        '$mercedes_img2',
        '$mercedes_img3',
        'Rustical Mercedes G Class for sale!',
        1, 'New', 'Brown', 'Automatic', 'Petrol',
        2017, 4, 5, 0, '1.6s', 285, 560, 65000,
        90000, now(), '2021-01-29 14:59:00'),
        ('BMW7 Series', 'boris.johnson@uk.com',
        '$bmw7_img1',
        '$bmw7_img2',
        '$bmw7_img3',
        'Exclusive BMW7 Series offered! Rarely used!',
        2, 'Used', 'Silver', 'Automatic', 'Petrol',
        2018, 4, 5, 2450, '1.2s', 300, 800, 90000,
        115000, now(), '2021-03-29 14:59:00');";
```

This query inserts some test data into the newly created “items” table. Overall, this query inserts three different test auctions into the database. It must take as reference the newly created test seller email and test category IDs, because the ‘items’ table is constrained by these foreign keys. Prior to the execution of this query, images must be derived from a specified directory, converted into binary large objects and stored into PHP variables.

4. Conceptual design – Entity-Relationship Diagram (ERD)

In order to model the core components of the database that are required to satisfy all of the functionalities of the auction website, the following Entity-Relationship Diagram (ERD) has been created.



In the next sections, the above ERD will be described in detail, and the assumptions about the processes that use the data will be presented.

4.1. Description

This part of the report includes the illustration of the design choices of the conceptual data model as well as brief descriptions of the data properties and the relationships between the entities.

Entities

Four separate entities that are involved in the processes of the auction website have been identified: buyer, seller, item (i.e. listing) and category (i.e. category of the item). Buyers and sellers have different roles and privileges within the website. Furthermore, the interaction between the seller and the buyer can be described as a typical customer-vendor relationship as they are engaged in interdependent transactions. They are therefore modelled as separate entities, which may interact with each other.

Items representing the listings are created by buyers and can have various attributes or features, setting the conditions of the auction and describing the product in more detail. Some of the attributes are optional and can have a value of NULL. Other attributes are mandatory and are therefore given a NOT NULL constraint. Some core attributes have been identified, which are essential for the functionality of the whole website. These include “itemID”, “startPrice”, “reservePrice”, “startDate” and “endDate”. The reserve price is essential for the auction process but is optional for the seller. The “itemID” is the primary key for the items table, which must be included as it provides a unique identifier for each listing.

The item category is used as the main feature to classify the items across various parts of the website and it plays a significant role in the buyer recommendations page. Thus, it has been isolated into an individual entity, where the “categoryID” is the primary key in the table, linking the individual categories with the items that they classify.

Relationships

When a seller logs into their account they are able to create an auction, by listing an item. Therefore, there exists a relationship between the seller and the items entity, whereby a seller can auction, or list, an item. Since the seller can list multiple items but an individual item can only have exactly one seller, the above multiplicity is given.

A buyer, however, can put multiple items on their watchlist as well as place multiple bids on items. Consequently, an item can be watched and receive bids by multiple buyers. In order to capture a bid properly, a bid price and bid date is also required. This is depicted by the relationship attributes.

After the time for the auction has elapsed, the transaction is triggered and the item is purchased. This is regarded as a different relationship than the bid, as it actively involves the two parties, buyer and seller, directly. The buyer has to transfer the set purchase price, whilst the seller is committed in return to hand the item over to the buyer. This indicates a multidimensional relationship, involving the three entities buyer, seller and item. In practice, this means that a seller can purchase multiple items from one buyer, resulting in the multiplicity illustrated above. Relevant attributes such as the purchase price and the purchase date can be derived from the bids and items entity (purchase price is the highest bid on the item and purchase date is the “endDate” of the item) and will not be included in the purchases table as it may interfere with the Third Normal Form. This is due to the fact that the “itemID” as a non-primary key would then functionally determine the purchase price and purchase date. Finally, the purchases entity is included in the ERD in order to clearly distinguish the purchased items.

4.2 Assumptions about processes that use the data

In order to be able to build a functioning website, certain assumptions about the processes that use the data have to be made, which decide how data is handled during specific events that occur. These requirements have to be consistent across the whole website and can be defined by the website owner individually.

In the create auction process, the seller must set the end date and time of the auction individually. Moreover, the amount of categories in the categories entity (table) is fixed at the beginning and cannot be altered by any procedure on the website. Consequently, the user must select one of the categories when creating a listing and cannot add their own category name.

For the bidding procedure, a bid will only be added to the database if it is 1 percent higher than the current highest bid price. This has been decided to avoid very small bid amounts. Additionally, there is no constraint for the number of successive bids by the same buyer on a single item. This means that a buyer can place a new bid on an item even if they are currently the highest bidder. However, if a person is registered as a buyer and as a seller in the system, they are not allowed to bid (when logged in as a buyer) on an item that they have listed themselves (when logged in as a seller). The reason for this is that a seller should not be able to “artificially” create demand for their own product by placing bids on it.

Whenever a buyer submits a valid bid and the item is not already in their watchlist, the item will be automatically added to it, since it is assumed that the buyer would want to be updated about the listing (for example by getting email notifications about new bids, etc.). Items can also be added and removed from the watchlist manually by the buyer.

The auction ends when the end date and time, which are initially set by the seller, are exceeded. A new row is then inserted into the purchases table. The new entry includes a “purchaseID” as a primary key (PK), which is automatically incremented and the “itemID” as a foreign key (FK). As mentioned above, the purchase date and the purchase price are derived from the items and the bids entities respectively, if they are required for specific features or processes. Moreover, the sold item will be removed from all the buyers’ watchlists automatically, as soon as the “endDate” elapses.

5. Translation of database schema

In the following sections, the translation procedure of the ERD will be elaborated starting with the translation of entities.

5.1 Translation of entities

First, the seller and buyer entities will be translated. These are strong entities, where the attributes of the ERD will be used as the columns, having the “email” column as a primary key for both entities. A “buyerID” or “sellerID” has not been used as the primary key, as having these as a primary key with a unique “email” column in the same table would mean that the tables would not be in Third Normal Form. This will be further elaborated in a designated section of the report.

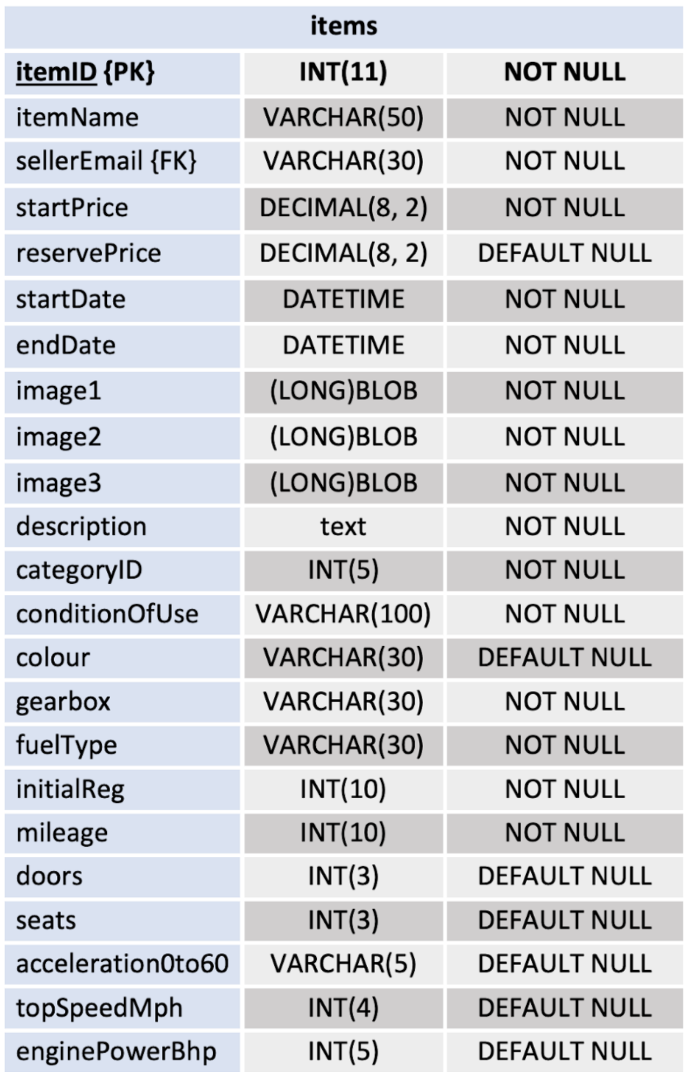
buyer		
email {PK}	**VARCHAR(30)**	**NOT NULL**
fName	VARCHAR(15)	NOT NULL
lName	VARCHAR(15)	NOT NULL
password	VARCHAR(40)	NOT NULL
phoneNo	VARCHAR(15)	DEFAULT NULL
street	VARCHAR(20)	DEFAULT NULL
city	VARCHAR(20)	DEFAULT NULL
Postcode	VARCHAR(10)	DEFAULT NULL

seller		
email {PK}	**VARCHAR(30)**	**NOT NULL**
fName	VARCHAR(15)	NOT NULL
lName	VARCHAR(15)	NOT NULL
password	VARCHAR(40)	NOT NULL
phoneNo	VARCHAR(15)	DEFAULT NULL
street	VARCHAR(20)	DEFAULT NULL
city	VARCHAR(20)	DEFAULT NULL
Postcode	VARCHAR(10)	DEFAULT NULL

The categories entity is also a strong entity and is translated into a table with the “categoryID” (automatically incremented) attribute as a primary key and with a “categoryNames” column.

category		
categoryID {PK}	**INT(2)**	**NOT NULL**
categoryName	VARCHAR(65)	NOT NULL

The last entity which has to be translated is the “items” entity. The items entity is considered a weak entity, because the existence of an item (or listing) within the website is dependent on the existence of both “seller” and “categories” entities. Hence, the items entity is translated to a table with a unique “itemID” as the primary key and a foreign key “sellerEmail” referencing the primary key “email” in the seller table. The “categoryID” is another foreign key in the new “items” table, referencing the “category” table. All the other attributes will be translated to individual columns with information about the auction process and the listed item.



items		
<u>itemID</u> {PK}	**INT(11)**	**NOT NULL**
itemName	VARCHAR(50)	NOT NULL
sellerEmail {FK}	VARCHAR(30)	NOT NULL
startPrice	DECIMAL(8, 2)	NOT NULL
reservePrice	DECIMAL(8, 2)	DEFAULT NULL
startDate	DATETIME	NOT NULL
endDate	DATETIME	NOT NULL
image1	(LONG)BLOB	NOT NULL
image2	(LONG)BLOB	NOT NULL
image3	(LONG)BLOB	NOT NULL
description	text	NOT NULL
categoryID	INT(5)	NOT NULL
conditionOfUse	VARCHAR(100)	NOT NULL
colour	VARCHAR(30)	DEFAULT NULL
gearbox	VARCHAR(30)	NOT NULL
fuelType	VARCHAR(30)	NOT NULL
initialReg	INT(10)	NOT NULL
mileage	INT(10)	NOT NULL
doors	INT(3)	DEFAULT NULL
seats	INT(3)	DEFAULT NULL
acceleration0to60	VARCHAR(5)	DEFAULT NULL
topSpeedMph	INT(4)	DEFAULT NULL
enginePowerBhp	INT(5)	DEFAULT NULL

5.2 Translation of relationships

Having translated the entities of the ERD, part of the relationships involving the different entities will be converted to tables.

The ERD shows that buyers can bid on items. As a buyer can bid on multiple items and an item can receive bids from multiple buyers, this binary relationship is a many-to-many (*:*) relationship. Thus, a dedicated table “bids” has been created for this relationship. The primary key is set to the unique “bidID” (automatically incremented) and the foreign keys “buyerEmail” and “itemID”, referencing the primary key email from the “buyer” table and the primary key of the “items” table respectively. The relationship attributes “bidPrice” and “bidDate” will be also included in the “bids” table.

bids		
bidID {PK}	**INT(11)**	**NOT NULL**
buyerEmail {FK}	VARCHAR(30)	NOT NULL
itemID {FK}	INT(11)	NOT NULL
bidPrice	DECIMAL(8, 2)	NOT NULL
bidDate	DATETIME	NOT NULL

Furthermore, buyers can watch items, by adding them to their watchlist. Again, this is a many-to-many (*:*) relationship and will be translated similarly to the “bids” relationship. A dedicated table for the watchlist, called “listings_watched” has been created. The foreign keys “buyerEmail” and “itemID”, referencing the primary keys in the “buyer” and “items” table, are again used. The primary key is the “watchID“ (automatically incremented). Here, no other attributes are required to model this relationship.

listings_watched		
watchID {PK}	**INT(11)**	**NOT NULL**
buyerEmail {FK}	VARCHAR(30)	NOT NULL
itemID {FK}	INT(11)	NOT NULL

Lastly, there has been defined a higher-order relationship of degree 3, involving the buyer, the seller and the items entity in the ERD. Consequently, a dedicated table “purchase” has been created for the relationship. Here, instead of using a composite primary key, consisting for instance of the “itemID” and the email of the buyer (e.g. “buyerEmail”), a new primary key “purchaseID” (automatically incremented) is defined for simplicity. Finally, only the “itemID” is included as a foreign key in the purchases table, as the “sellerEmail” and the “buyerEmail” would be functionally dependent on the non-PK attribute “itemID”. This would eventually cause the table to not be in 3NF, thus the email of the seller and the buyer are excluded.

purchase		
purchaseID {PK}	**INT(11)**	**NOT NULL**
itemID {FK}	INT(11)	NOT NULL

6. Analysis of Database Schema (Third Normal Form)

The third normal form (3NF) ensures that all non-primary key attributes can only be determined by the primary key.

The “auction” database contains the tables “bids”, “buyer”, “category”, “items”, “listings_watched”, “purchase” and “seller”.

- ➔ bids (<u>bidID </u>(PK), buyerEmail (FK), itemID (FK), bidPrice, bidDate)

The bids table is in third normal form, as the “bidID” (PK) functionally determines all the other attributes of the table. The non-primary key attributes are independent from each other. An item can receive bids from multiple buyers, one buyer can bid on multiple items etc. The bid price and date can also vary but can only be determined by the primary key.

- ➔ buyer (<u>email</u> (PK), fName, lName, password, phoneNo, street, city, postcode)

The buyer table is in third normal form. The primary key “email” is a unique identifier of the buyer’s personal information, because one email is always associated with one buyer. This table describes the personal information of the buyer. The non-primary key attributes cannot functionally determine each other because multiple people can share the same names, passwords, input the same phone numbers, live in the same city etc. There is no variable that determines one of these attributes other than the unique email of the buyer.

- ➔ seller (<u>email</u> (PK), fName, lName, email, password, phoneNo, street, city, postcode)

The seller table has the same attributes as the buyer table. Therefore, the seller table is also in third normal form.

- ➔ category (<u>categoryID</u> (PK), categoryName)

The category table is in third normal form, as its only non-primary key attribute, “categoryName”, is functionally determined by the primary key, “categoryID”.

- ➔ items (<u>itemID</u> (PK), itemName, sellerName (FK), image1, image2, image3, description, categoryID(FK), conditionOfUse, colour, gearbox, fuelType, initialReg, doors, seats, mileage, accelaration0to60mph, enginePowerBhp, startPrice, reservePrice, startDate, endDate)

The items table is in third normal form, as all the non-primary key attributes are main features that only depend on the item. Thus, only the “itemID” determines its attributes. The non-primary key attributes are completely independent from each other. The starting price of an item does not depend on one of the images uploaded by the seller, the gearbox does not depend on the engine power etc.

- ➔ listings_watched (<u>watchID</u> (PK), buyerEmail (FK), itemID (FK))

The listings_watched table is in third normal form, as the non-primary key attributes can only be determined by the “watchID”. The “itemID” cannot be determined by the “buyerEmail”, as one item might receive bids from multiple buyers. The “buyerEmail” can also not be determined by the “itemID”, as one buyer can bid on multiple items. The primary key is the only attribute that functionally determines all the other attributes.

➔ purchase (<u>purchaseID</u> (PK), itemID (FK))

The purchase table is in third normal form, as its only non-primary key attribute, “itemID” is functionally determined by the primary key, “purchaseID”.

The “auction” database is therefore in third normal form, as the primary keys of all tables in the database uniquely determine all other non-primary key attributes.

7. Explanation of Database Queries

This section of the report will explain each of the MySQL database queries that are used within the various PHP webpages for the auction website.

process_registration.php

```
$sql = "SELECT email FROM buyer WHERE email = ? ;";
$sql_2 ="SELECT email FROM seller WHERE email = ? ;";
```

The purpose of the first MySQL query is to check the existence of the email used to sign up. The query selects the email equal to the input email address from either the buyer or seller table (depending on the account type). If this query has returned any result, it means that the sign-up email already exists.

```
if ($user['accountType'] == "buyer") {
        $query = "INSERT INTO buyer (fName, lName, email, password, phoneNo, s
treet, city, postcode)
              VALUES (?, ?, ?, SHA(?), ?, ?, ?, ?)";
    } elseif ($user['accountType'] == "seller") {
        $query = "INSERT INTO seller (fName, lName, email, password, phoneNo,
street, city, postcode)
                VALUES (?, ?, ?, SHA(?), ?, ?, ?, ?)";
    } else {
        $query = "INSERT INTO buyer (fName, lName, email, password, phoneNo, s
treet, city, postcode)
              VALUES (?, ?, ?, SHA(?), ?, ?, ?, ?);";

        $query_2 ="INSERT INTO seller (fName, lName, email, password, phoneNo,
 street, city, postcode)
              VALUES (?, ?, ?, SHA(?), ?, ?, ?, ?)";
    }
```

The purpose of this next query is to save buyer or seller information when they sign up. This query inserts the following fields into the buyer or seller table, depending on the account type: buyer or seller email (a mandatory field as the email is the primary key), the mandatory fields first name, last name, password (with a SHA function), and the optional fields telephone number, street, city and postcode.

create_auction_result.php

```
  $catsql = "INSERT INTO category (`categoryName`)
                SELECT * FROM (SELECT '$category') as tmp
                WHERE NOT EXISTS (
                    SELECT `categoryName`
                    FROM category
                    WHERE `categoryName` = '$category');";
```

This query inserts the category name selected by the user in the dropdown menu of the ‘Create auction’ page into the category table in the database in case it does not already exist there.

```
$categoryID = "SELECT `categoryID`
                    FROM category
                    WHERE `categoryName` = '$category';";
```

After the category name is inserted into the database, it gets assigned a related ‘categoryID’ with auto increment. We run this query to select the related ‘categoryID’ and assign it to a PHP variable which we are going to use later.

```
  $sql = "INSERT INTO items (
        `itemName`, `sellerEmail`, `image1`, `image2`, `image3`,
        `description`, `categoryID`, `conditionOfUse`, `colour`,
        `gearbox`, `fuelType`, `initialReg`, `doors`, `seats`,
        `mileage`, `acceleration0to60mph`,`topSpeedMph`,
        `enginePowerBhp`, `startPrice`, `reservePrice`,
         `startDate`, `endDate`)
         VALUES ('$itemName', '$sellerEmail', '$image_1',
         '$image_2', '$image_3', '$description', $categoryID,
         '$conditionOfUse', '$colour', '$gearbox', '$fueltype',
         $initialreg, $doors, $seats, $mileage, '$accelaration',
         $topspeed, $enginepwr, $startPrice, $reservePrice, now(),
         '$endDate');";
```

Upon successful creation of an auction, this query takes as values all item features defined by the seller in the ‘Create auction’ page and inserts them into the respective columns of the ‘items’ table in the database. The ‘sellerEmail’ is retrieved by the user’s session, assigned to a PHP variable and then inserted into the database. The ‘startDate’ column takes as value the SQL function now(), as the auction starts as soon as the auction is created by the seller.

listing.php

```
$sql = "SELECT * FROM items WHERE itemID = $item_id;"
```

This query selects all the columns for a particular item.

```
$sql = "SELECT MAX(bidPrice) AS bidPrice
        FROM bids
        WHERE itemID = $item_id;";
```

This query gets the highest bid price for a particular item.

```
$sql = "SELECT COUNT(bidID) AS bidCount
        FROM bids
        WHERE itemID = $item_id;"
```
This query counts the number of bids, that have been placed on a particular item.

```
$sql = "SELECT c.categoryName as category
        FROM category c, items i
        WHERE (c.categoryID = i.categoryID AND i.itemID = $item_id);";
```
This query outputs the categoryName column from the category table for a particular item from the items table.

```
$sql = "SELECT COUNT(watchID) AS watchCount
        FROM listings_watched
        WHERE itemID = $item_id;";
```
This query counts the number of times a particular item has been added to the watchlist, i.e., how many users are watching the particular item, since a user can add an item to their watchlist only once.

```
$sql = "SELECT buyerEmail
        FROM bids
        WHERE itemID = $item_id
        AND bidPrice = (SELECT MAX(bidPrice)
                        FROM bids
                        WHERE itemID = $item_id);";
```
This query gets the buyer (i.e. buyerEmail, which is a unique identifier for the buyer), who is currently the highest bidder for a particular item.

```
$sql = "SELECT COUNT(watchID) as wCount
        FROM listings_watched
        WHERE buyerEmail = '$buyerID'
        AND itemID = $item_id;";
```
This query checks if a particular buyer is watching a specific item, by counting the watchIDs with the given conditions. This should return 1 if the buyer is watching and 0 if not.

```
$sql = "SELECT COUNT(p.purchaseID) AS pCount
        FROM purchase p
        WHERE itemID=$item_id;";
```
This query checks if a particular item has been purchased, by counting the purchaseIDs with the given condition. This should return 1 if the item has been purchased and 0 if not.

```
$sql = "SELECT b.bidPrice AS purchasePrice
        FROM purchase p
        INNER JOIN items i ON p.itemID = i.itemID
        INNER JOIN bids b ON i.itemID = b.itemID
        WHERE b.itemID=$item_id
        AND b.bidPrice = (SELECT MAX(bidPrice)
                          FROM bids
                          WHERE itemID = $item_id);";
```

This query gets the purchase price by selecting the maximum bid price from the bids table for the specific item. To do this, three tables have to be joined together: The items table is joined on the purchase table and the bids table is joined on the items table.

browse.php

```
$sql_query_category = "SELECT categoryID, categoryName FROM category;";
```

The purpose of this query is to show all categories in the dropdown list. This query gets all categories from the category table.

```
 $query_condition = "";

  if ($keyword != "") {
      $query_condition .= " AND itemName like '%" . $keyword . "%' ";
  }

  if ($category != "" and $category != "all") {
      $query_condition .= " AND categoryID = " . $category . " ";
  }

  $query_ordering = "";
  if ($ordering == "" or $ordering == "pricelow") {
      $query_ordering .= " ORDER BY currentPrice ";
  } elseif ($ordering == "pricehigh") {
      $query_ordering .= " ORDER BY currentPrice DESC ";
  } elseif ($ordering == "date") {
      $query_ordering .= " ORDER BY enddate ";
  }
$sql_query_items = "SELECT i.itemID, itemName, description, categoryID, IFNULL
(b.numBid, 0) as numBid, IFNULL(b.currentPrice, i.startPrice) as currentPrice,
 DATE_FORMAT(endDate, '%Y-%m-%dT%H:%i:%s') as endDate
                      FROM items as i
                      LEFT JOIN (SELECT itemID ,count(*) as numBid, Max(bidPri
ce) as currentPrice FROM bids GROUP BY itemID) as b
                      On i.itemID = b.itemID
                      WHERE NOW() < endDate " . $query_condition . $query_orde
ring;
```

The purpose of this query is to get the list of auction items which are still active and show their names, descriptions, end times and the prices. If no one placed any bids then show the item starting price otherwise show the maximum bid price. The $query_condition and $query_ordering variables will be used to store the search bar user input information.

```
$results_per_page = 10;
$start_record = ($curr_page - 1) * $results_per_page;
$max_page = ceil(mysqli_num_rows($result_pagination) / $results_per_page);
$query_for_each_page = $sql_query_items . " LIMIT $start_record, $results_per_
page";
```

For the pagination, it sets results per page = 10 records and uses (current page - 1) * results per page to constrain the number of rows returned by this query (LIMIT clause).

mylistings.php

```
   $query = "SELECT i.itemID, itemName, description, categoryID,
                    IFNULL(b.numBid, 0) as numBid,
                    IFNULL(b.currentPrice, i.startPrice) as currentPrice,
                    DATE_FORMAT(endDate, '%Y-%m-%dT%H:%i:%s') as endDate
              FROM items as i
              LEFT JOIN (SELECT itemID ,count(*) as numBid,
                            Max(bidPrice) as currentPrice
                            FROM bids GROUP BY itemID) as b
              On i.itemID = b.itemID
              WHERE sellerEmail = '$id' ";
```

The initial MySQL query used in this page selects the item ID, name, description, category ID, the number of bids, the current price and the end date of items that are listed by the current logged in seller.

Item ID, name, description and category ID of each item are extracted from the items table. “numBid”, “currentPrice” and “endDate” are selected from the bids table. The “numBid” column is formatted so if there are currently no bids on an item, this will appear as zero. The current price column is formatted so that if there are no bids on an item, the price will appear as the starting price.

The bids table is joined to the items table using a left join to preserve the rows in the newly created table. “GROUP BY itemID” is used to group all of the information by item. The “numBid” variable is created by counting all of the data and therefore the number of bids that are in the database for each item. The current price is calculated by using “Max(bidPrice)” for each item which is the maximum price that is currently bid on each item.

A “WHERE” clause is used to filter for items that are only listed by the seller that is currently logged in.

```
$query_for_each_page = $query . " LIMIT $start_record, $results_per_page";
```

This query is used to set up the pagination, by limiting the results per page to 10, ranging from the first record.

```
$expireditems = "SELECT itemID FROM items
                 WHERE NOW() > endDate AND sellerEmail = '$id' ";
```

This query extracts inactive items from the items table that are listed by the seller. The resulting item IDs are inserted into an array, then a function can be called to display these unsold items with a banner.

```
 $sql = "SELECT p.itemID AS itemID
            FROM purchase p
            INNER JOIN bids b ON p.itemID = b.itemID
            INNER JOIN items i ON b.itemID = i.itemID
            WHERE i.sellerEmail = '$id';";
```

A third MySQL query selects sold items that are listed by the seller to input this information into another array, where a function can be called so that sold items have a banner displayed to the seller. This query uses two inner joins to join the purchase, bids and items tables.

mybids.php

```
$query = "SELECT i.itemID, itemName, description, categoryID,
                     DATE_FORMAT(endDate, '%Y-%m-%dT%H:%i:%s') as endDate,
                     IFNULL(c.numBid, 0) as numBid,
                     c.currentPrice, b.buyerEmail
              FROM items as i
              JOIN bids as b
              On i.itemID = b.itemID
              LEFT JOIN(SELECT itemID, count(*) as numBid,
                          max(bidPrice) as currentPrice
                          FROM bids GROUP BY itemID) as c
              On b.itemID = c.itemID
              WHERE b.buyerEmail = '$id' AND NOW() < endDate GROUP BY itemID";
```

The initial MySQL query used in this page selects the item ID, name, description, category ID, the number of bids, the current price and the end date of items from the items and bids tables that have been bid on by the current logged in buyer. It uses a very similar query to that of the mylistings.php page.

All of the information from the bids table is joined using an inner join (“JOIN”) clause so that the buyer ID for each bid can be extracted. This selects records that has matching values in both tables.

A left join (“LEFT JOIN”) is used to join the selected information from the items and bids tables with calculated variables from the bids tables – “numBid” and “currentPrice”. A nested “SELECT” query is used within this to select the relevant information from the bids table. “GROUP BY itemID as c” is used to group all of the information from the bids table by item.

A “WHERE” clause is used to filter for items that are currently active and that have been bid on by the logged in buyer. “GROUP BY” is used to specify that the bids should be grouped by item ID so that each auction that is pulled up is a different item and repeat bids are not displayed to prevent the page from becoming cluttered.

```
$query_for_each_page = $query . " LIMIT $start_record, $results_per_page";
```
This query is used to set up the pagination, by limiting the results per page to 10, ranging from the first record.

mypurchases.php

```
 $query = "SELECT i.itemID, itemName, description, categoryID,
                  DATE_FORMAT(endDate, '%Y-%m-%dT%H:%i:%s') as endDate,
                  IFNULL(c.numBid, 0) as numBid,
                  c.currentPrice, b.buyerEmail
              FROM items as i
              JOIN bids as b
              On i.itemID = b.itemID
              LEFT JOIN(SELECT itemID, count(*) as numBid,
                           max(bidPrice) as currentPrice
                           FROM bids GROUP BY itemID) as c
              On b.itemID = c.itemID
              JOIN purchase as p
              ON p.itemID=i.itemID
             WHERE b.buyerEmail = '$id' GROUP BY itemID ";
```
This query selects item ID, item name, description, category ID, end date, number of bids and current price of items that have been purchased by the current logged in buyer. The query is nearly identical to that of the MyBids page, except the purchase table is also joined so that the resulting output table only contains items that have been purchased by the buyer, filtered for using the “WHERE” clause.

```
$query_for_each_page = $query . " LIMIT $start_record, $results_per_page";
```
This query is used to set up the pagination, by limiting the results per page to 10, ranging from the first record.

```
$sql = "SELECT i.itemID AS itemID
            FROM purchase p
            INNER JOIN bids b ON p.itemID = b.itemID
            INNER JOIN items i ON b.itemID = i.itemID
            WHERE b.buyerEmail = '$id';";
```
This query selects items that have been purchased by the buyer to input this information into another array, where a function can be called so that sold items have a banner displayed to the buyer. This query uses two inner joins to join the purchase, bids and items tables. The resulting table includes all of the items that have been purchased. The “WHERE” clause filters for items that have been purchased by the buyer that is currently logged in.

place_bid.php

```
$sql = "SELECT sellerEmail
        FROM items
        WHERE itemID = $item_id;";
```
This query returns the seller email for the seller who has listed a particular item.

bid_result.php

```
$sql = "SELECT buyerEmail
        FROM bids
        WHERE itemId = $item_id
        AND bidPrice = (SELECT MAX(bidPrice)
                        FROM bids
                        WHERE itemID = $item_id)";
```

This query gets the buyer (i.e. buyerEmail, which is a unique identifier for the buyer), who is currently the highest bidder for a particular item.

```
$sql = "SELECT email, CONCAT(fName,' ', lName) as username
        FROM buyer
        WHERE email = '$highest_bidder';";
```

This query selects the email and the concatenated first and last name of the buyer whose email (PK) matches the highest bidder (delimited with a whitespace).

```
$sql = "SELECT b.email, CONCAT(b.fName,' ', b.lName) AS username
        FROM listings_watched lw
        INNER JOIN buyer b ON b.email = lw.buyerEmail
        WHERE lw.itemID = $item_id;";
```

This query gets the email and the concatenated first and last name of the buyers that watch a particular item.

```
$sql = "SELECT COUNT(1) as countTrue
        FROM listings_watched
        WHERE buyerEmail = '$buyer_id'
        AND itemID = $item_id;";
```

This query returns 1 (true) if a particular buyer watches a specific item.

```
$sql = "INSERT INTO listings_watched(`buyerEmail`, `itemID`)
        VALUES ('$buyer_id', $item_id);";
```

This query inserts the buyer email and the itemID into the listings_watched table.

```
$sql = "INSERT INTO bids (`buyerEmail`, `itemID`, `bidPrice`, `bidDate`)
        VALUES ('$buyer_id', $item_id, $new_bid_price, now());";
```

This query inserts the buyer email, the itemID, the new (highest) bid price and the bid datetime (now) into the bids table.

timer_event.php

```
SELECT org.itemID,items.itemName, items.sellerEmail, CONCAT(seller.fName, ' ',
 seller.lName) As SellerName,
    bidID, org.buyerEmail, CONCAT(buyer.fName, ' ', buyer.lName) As BuyerName,
 maxPrice.price AS buyerBidPrice
FROM bids as org
INNER JOIN
 (SELECT itemID, max(bidPrice)as price
  FROM bids
  WHERE itemID in (SELECT itemID FROM items
           WHERE NOW() > endDate
           AND itemID NOT IN (SELECT itemID FROM purchase))
  GROUP BY itemID) as maxPrice
ON org.itemID = maxprice.itemID and org.bidPrice = maxprice.price
INNER JOIN items
ON org.itemID = items.itemID and maxprice.price >= items.reservePrice
INNER JOIN buyer
ON buyer.email = org.buyerEmail
INNER JOIN seller
ON seller.email = items.sellerEmail;
```

This query gets the expired auctions' outcome and inserts it into a temporary table, which is defined by a current DateTime greater than endDate, and items that are not in purchase table.The columns include itemID (of the item that was sold), item name, seller email, seller name, buyer’s email (the winner), buyer name, winner's bid ID, sold price (where maximum bid price is greater than reserve price) and end date of the auction.

```
INSERT INTO purchase(itemID)
SELECT itemID FROM temp_auction_outcome;
```

This query inserts the result into the purchase table in order to save the item that has been sold.

```
CREATE TEMPORARY TABLE temp_remove_lw
SELECT items.itemID, items.itemName, lw.buyerEmail, CONCAT(buyer.fName, ' ', b
uyer.lName) AS name
FROM items
JOIN listings_watched AS lw
ON lw.itemID = items.itemID
JOIN buyer
ON buyer.email = lw.buyerEmail
WHERE NOW() > items.endDate;
```

This query gets expired items which are still in the watchlist and inserts them into a temporary table.

```
DELETE FROM listings_watched
WHERE itemID IN (SELECT DISTINCT itemID FROM temp_remove_lw);
```

This query deletes temp_remove_lw records.

```
SELECT 'timer_event', temp_auction_outcome.* FROM temp_auction_outcome;
```
This query selects information of the winner of the item in order to send the email announcement that states that they won the item. It also selects the seller's information of the item sold in order to send the email announcement that their item has been sold.

```
SELECT 'itemClosed', temp_remove_lw.*
FROM temp_remove_lw
WHERE temp_remove_lw.buyerEmail NOT IN (select buyerEmail from temp_auction_ou
tcome);
```
This query selects the information of the buyer who had added the expired items into their watch list but did not win the item, in order to send the email announcement that the sale of the item has been closed and it has been sold to someone else.

```
DROP TEMPORARY TABLE temp_auction_outcome;
DROP TEMPORARY TABLE temp_remove_lw;";
```
This query drops both of the temporary tables.

watchlist_func.php

```
$sql = "SELECT itemName
        FROM items
        WHERE itemId=$item_id";
```
This query returns the item name/title of a particular item from the items table.

```
$sql = "INSERT INTO listings_watched(`buyerEmail`, `itemID`)
          VALUES ('$buyerID', $item_id);";
```
This query inserts the buyer email and the itemID into the listings_watched table.

```
$sql = "DELETE FROM listings_watched
        WHERE buyerEmail='$buyerID'
        AND itemID=$item_id;";
```
This query removes a particular item from the watchlist (listings_watched table) of a specific buyer.

recommendations.php

```
 $numbid_curr_price_per_item = "SELECT itemID, count(*) as numBid,
                                           Max(bidPrice) as currentPrice
                                       FROM bids GROUP BY itemID";
```
This query selects all the items of the bid table, counts all bids per item and selects the current price (the maximum price) for each item. This query will be used as a subquery for a bigger query later.

```
 //Select all items of a specific buyer
$items_of_buyer ="SELECT itemID FROM bids WHERE buyerEmail = '$buyerID'";
```
This query selects all the items a specific buyer has bid on. The specific buyer ID, their email, is retrieved by the user's session and then assigned to a PHP variable, which

is used in the WHERE clause. This query will be used as a subquery for a bigger query later.

```
//Select all categories that the specific buyer has bid on.
  $categories_bid_by_spec_buyer = "SELECT c.categoryID FROM bids b
                                    INNER JOIN items c ON c.itemID = b.itemID
                                    WHERE b.buyerEmail = '$buyerID'";
```

This query selects all the categories the specific buyer has bid on, by joining the bid and category tables. The specific buyer ID (email) is, again, retrieved by the user's session, assigned to a PHP variable and used in the WHERE clause to select only buyer-specific data. We will use this query as a subquery for a bigger subquery later.

```
//Select all buyers that have bid on the same categories as the specific buyer
  $buyers_same_categories = "SELECT b.buyerEmail FROM bids b
                 LEFT JOIN items c ON c.itemID = b.itemID
                 WHERE c.categoryID in (".$categories_bid_by_spec_buyer.")";
```

This query selects all buyers that have bid on the same categories as the specific buyer. A left join joins the category ID to the bids table. Then, the WHERE IN clause filters for categories that also appear in the 'categories_bid_by_spec_buyer' subquery. As a result, all buyers with similar category preferences as the specific buyer will be listed. We will use this query as a subquery for a bigger subquery later.

```
//Select all items that all buyers, who have bid on the same categories as the
 specific buyer, have bid on
$recommended_items = "SELECT itemID
                      FROM bids
                      WHERE buyerEmail in (".$buyers_same_categories.")";
```

This query selects all items that all buyers, who have bid on the same categories as the specific buyer, have bid on. The WHERE IN clause filters for the buyers that also appear in the 'buyers_same_categories' subquery. As a result, all items that have had bids from buyers with similar category preferences as the specific buyer will be listed. We will use this query as a subquery for a bigger query later.

```
  //Main SQL query, containing all the relevant subqueries and listing all the
 recommended items for the buyer.
  $main_query_items = "SELECT i.itemID, itemName, description, i.categoryID, b
.numBid,  b.currentPrice, DATE_FORMAT(endDate, '%Y-%m-%dT%H:%i:%s') as endDate
                        FROM items as i
                        LEFT JOIN (".$numbid_curr_price_per_item.") b
                      ON i.itemID = b.itemID
                      WHERE now() < endDate and numBid > 0 and i.itemID NOT IN
 (".$items_of_buyer.") and i.itemID IN (".$recommended_items.")
                      ORDER BY endDate";
```

This is the main query that combines all the subqueries listed above in order to select item ID, name, description, category ID, the number of bids, the current price and the end date of items that might be of special interest to an individual buyer.

Item ID, item name, description, category and end date are selected from the items table. The number of bids and the current price per item are selected as table ‘b’ from the ‘numbid_curr_price_per_item’ subquery and left joined to items i.

The WHERE clause filters for only active items and items that have already had at least one bid. Furthermore, the WHERE clause filters for items the specific buyer has not yet bid on, by using the NOT IN operator in connection with the ‘items_of_buyer’ subquery. Finally, the WHERE clause filters for items that other buyers, who have bid on similar categories as the specific buyer, are currently bidding on, by using the IN operator in connection with the ‘recommended_items’ subquery.

The ORDER BY clause sorts the items by end date. The most urgent auctions will appear first in the list.

```
//SQL query, counting the number of the recommended items to determine the
pagination
  $sql = "SELECT COUNT(*) as count FROM (".$main_query_items.") t ";
```

This query counts all the entries of the ‘main_query_items’ table. This is done to calculate the maximum number of pages for the pagination.

```
//This query will show up to 10 recommendations per page.
  $sql_query_items_page = $main_query_items . " LIMIT $start_record, $results_
per_page";
```

This query shows up to 10 recommendations form the ‘main_query_items’ table per page. The limit range is determined by the PHP variables ‘start_record’ and ‘results_per_page’.

utilities.php

```
$sql = "";
    if ($account_type == "buyer") {
        $sql = "SELECT email as buyerID, CONCAT(fName,' ', lName) as username,
 email FROM buyer WHERE email = ? AND password = SHA(?)";
    } else {
        $sql = "SELECT email as sellerID, CONCAT(fName,' ', lName) as username
, email FROM seller WHERE email = ? AND password = SHA(?)";
    }
```

The purpose of this query is to check the validity of login information. This query uses user input email and password as a condition to query if the email and password match records in either buyer or seller table. If the record exists, then it will return the account typeID (depending on which type the user choses on the login page, could be either buyer or seller), first name + last name and email. The return value will be saved into SESSION.