TEAM 15 - Auction App Database Project

Video Demonstration Link

<https://mediacentral.ucl.ac.uk/Play/17206>

Username and password for email account used to send notifications:

username : databasecoursework@gmail.com

password : pFKdcJ4LxMNN8q7n

Entity relationship diagram & database schema

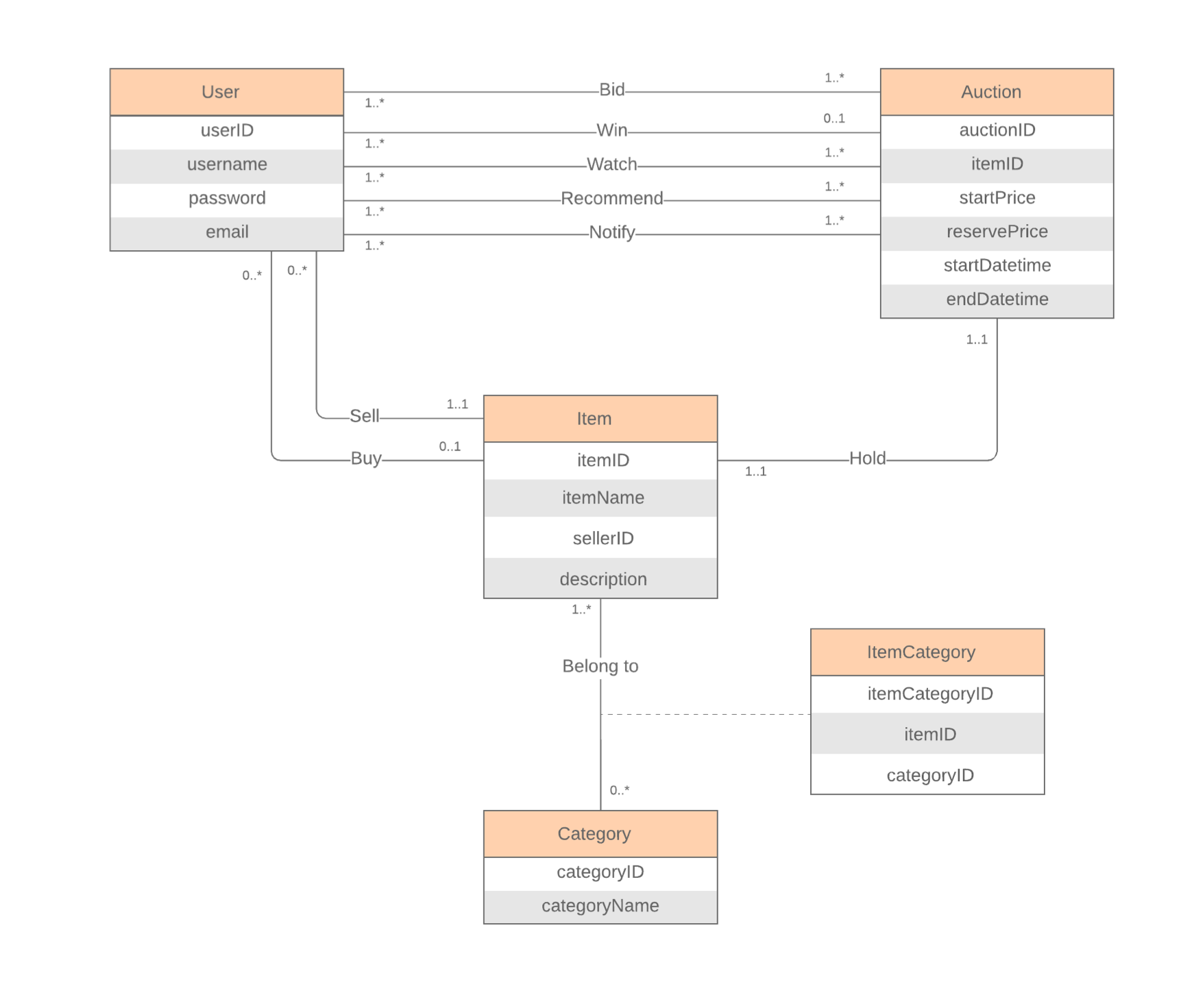
The bids table stores the bids that users make on particular auctions as well as the ID of the user who has made each bid, the bid amount and the ID of the auction that they bid on, where userID and auctionID are foreign keys. A user can also watch an auction and their information is stored in the ‘watchList’ table, where ‘userID’ and ‘auctionID’ are foreign keys. If a user wins an auction, the ‘buyerID’ and the ‘auctionID’ will be stored in the ‘purchaseHistory’ table. An auction (or item) can be recommended to users and the information is saved in the recommendations table, where the ‘userID’ and item name of the recommendations table are the foreign keys (the process in which items are recommended to users is detailed in the demo video).

Users get email notifications during an auction if they are outbid and

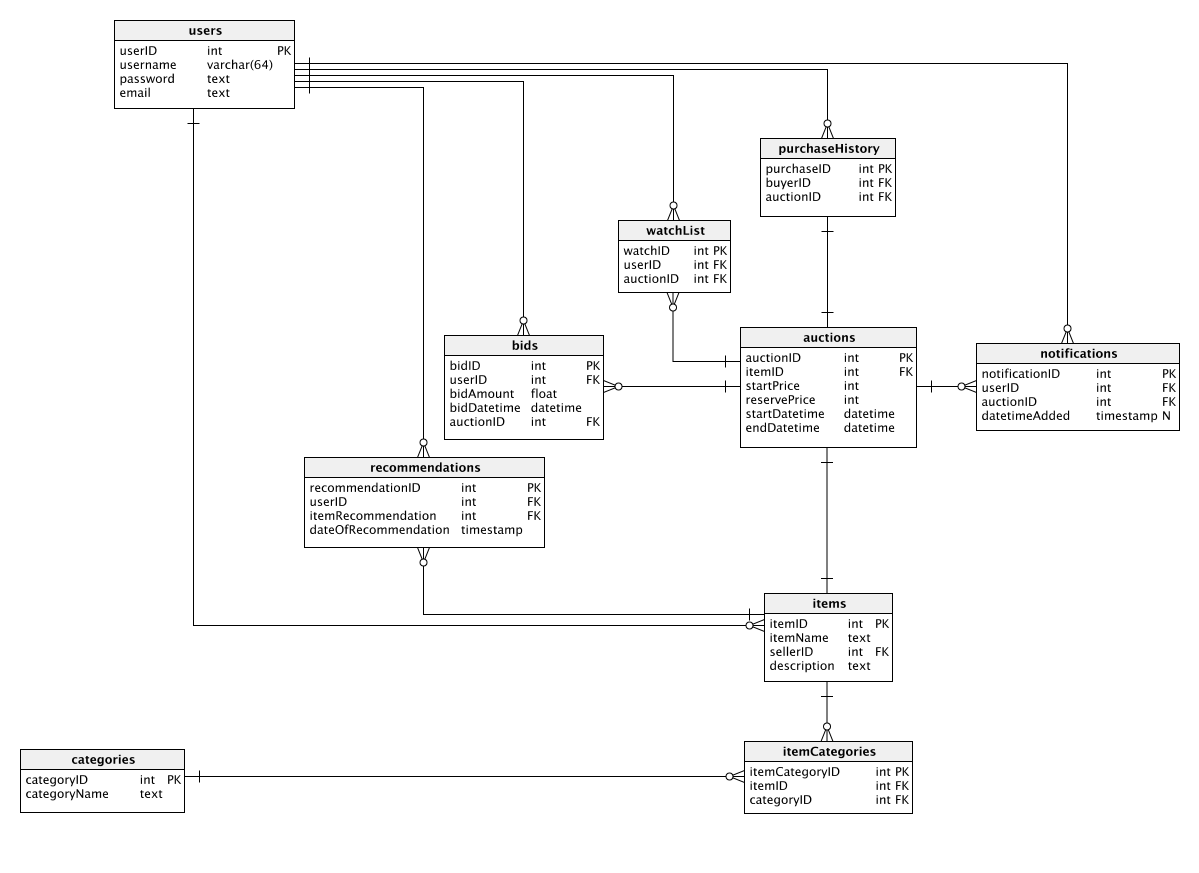
when an auction closes, if they are the highest bidder (and bid above the reserve price). The seller is also notified if their items sell (i.e. if at least one bid for that item exists and the highest bid amount is greater than the reserve price for the item). The notifications are added to the notifications table as they are generated and emailed out, where the ‘userID’ and the ‘auctionID’ are used as foreign keys.

A user can sell or buy items, which are stored in the items table. The items and the auctions tables are connected with itemID and have a 1 to 1 relationship, as each item must have one associated auction and vice versa. An item can also have one or more categories assigned to it and, to prevent transitive dependencies, the items and categories table are split, but still connected via the ‘itemCategories’ table.

ERD Diagram



Database Schema Diagram



Third Normal Form Analysis

Our database is in first normal form if it satisfies the following conditions[1]:

* If it only contains atomic values, meaning the data in each cell is singular and cannot be divided any further.
* If it only contains non-repeating groups, so data is not repeated in individual cells and columns containing the same data are not repeated.

Our schema satisfies the above conditions, and is therefore in 1st normal form.

A database is in second normal form if it satisfies the following conditions[1]:

* If it is in first normal form (see above).
* All the non-key attributes are fully functional dependent on the primary key. Therefore, tables that only contain a single key as their primary key are automatically in second normal form because there can be no partial dependencies.

Our schema satisfies the above conditions, and is therefore in 2nd normal form.

A database is in third normal form if it satisfies the following conditions[1]:

* If it is in second normal form (see above)
* If there are no transitive functional dependencies, so if an element is dependent on another and a third element is dependent on the first, we can say that the third is transitively dependent on the first.

Every attribute in each table of our schema if fully dependant on the primary key associated with that table; therefore our schema meets the requirements of 3rd normal form.

Database Queries

All interactions with the database can be found in the dbHelper.php file, and are also included here for convenience

**Fetch\_user:**

**"SELECT \* FROM users WHERE username = ?"**

This query returns all of the details for a user, which is selected using their username. This is also used to check if a user exists in the table, as it returns nothing if they do not.

**Insert\_user:**

**"INSERT INTO users (username, password, email) VALUES (?, ?, ?)”**

This query inserts a new user into the users table.

**Fetch\_future\_auctions\_by\_user**

**SELECT DISTINCT(bids.auctionID)**

**FROM bids, auctions**

**WHERE userID = ?**

**AND bids.auctionID = auctions.auctionID**

**AND auctions.endDateTime > now()**

This query returns all of the auctionIDs that the user has bid on for auctions that are still open. A DISTINCT condition is used to prevent duplicate IDs if the the user has bid on an item more than once.

**Fetch\_listing\_by\_user\_auction**

**SELECT i.itemName, i.description, b.bidAmount AS yourBid, b.bidDatetime AS yourBiddt, a.endDatetime**

**FROM items AS i, auctions AS a, bids AS b**

**WHERE b.userID = ?**

**AND b.auctionID = ?**

**AND i.itemID = a.itemID**

**AND a.auctionID = b.auctionID**

**ORDER BY yourBid DESC**

This query is used to populate the myBids table that shows a given user the items that they have bid on. This is done for a single auction, so the query needs to be executed several times in a loop for several auctionIDs.

**fetch\_max\_bid\_for\_auction**

**"SELECT bidAmount AS highestBid, bidDatetime AS highestBiddt**

**FROM bids WHERE auctionID = ?**

**ORDER BY highestBid DESC**

**LIMIT 1"**

This query returns the current highest bid for a given auction.

**fetch\_purchase\_history()**

**SELECT p.auctionID, i.itemName, i.description, a.endDatetime as purchaseDate**

**FROM items as i, auctions as a, purchaseHistory as p**

**WHERE p.auctionID = a.auctionID**

**AND a.itemID = i.itemID**

**AND p.buyerID = ?**

This query is used to retrieve the data required to populate the purchase history page using a join between the purchase history table, auctions table and items table. A purchase history table was used to avoid expensive and complicated operations on the bids table to find who won an auction after it has finished.

**Fetch\_sales\_history**

**SELECT p.auctionID, i.itemName, i.description, a.endDatetime as saleDate**

**FROM items as i, auctions as a, purchaseHistory as p**

**WHERE p.auctionID = a.auctionID**

**AND a.itemID = i.itemID**

**AND i.sellerID = ?**

This query is used to get all of the auction and item details required to populate the sales history table from the purchase history table. The returned rows are limited to the ones where the sellerID is equal to the input userID.

**Fetch\_your\_listing**

**Query 1**

**SELECT a.auctionID, i.itemName, i.description, a.endDatetime**

**FROM items as i, auctions as a**

**WHERE i.itemID = a.itemID**

**AND a.endDatetime > now()**

**AND i.sellerID = ?**

**GROUP BY a.auctionID, i.itemName, i.description, a.endDatetime**

**Query 2**

**SELECT COUNT(b.bidID) AS bidsNumber**

**FROM bids as b**

**WHERE auctionID = ?**

These queries are used to generate the table on the “Your listings” page. Query 1 gets almost all of the auctions details for each item listed by the given seller. Query 2 is then used to find the number of bids for each auction returned in query 1 and it is appended onto each row.

**Fetch\_user\_id\_from\_username**

**SELECT userID**

**FROM users**

**WHERE username = ?**

This query returns the auto-generated userID for a given username.

**Fetch\_watch\_list**

**SELECT a.auctionID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime**

**FROM watchList as wl, items as i, auctions as a**

**WHERE wl.auctionID = a.auctionID**

**AND a.itemID = i.itemID**

**AND a.endDatetime > now()**

**AND wl.userID = ?**

This is the query used to populate the watchlist page for each user. This query returns the auctionID (which is used to construct the hyperlink to the item auction page) and all the other item and auction details. The line a.endDatetime > now() is used so that a user only sees items that have not ended.

**Fetch\_item\_auction :**

**Query 1**

**SELECT i.itemID, i.sellerID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime, COUNT(b.bidID) AS bidsNumber**

**FROM items as i, bids as b, auctions as a**

**WHERE i.itemID = a.itemID**

**AND a.auctionID = b.auctionID**

**AND a.auctionID = ?**

**GROUP BY i.itemID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime**

**Query 2**

**SELECT i.itemID, i.sellerID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime**

**FROM items as i, auctions as a**

**WHERE i.itemID = a.itemID**

**AND a.auctionID = ?**

These queries are used in a function to fetch the details for a particular item and its auction so that they can be displayed on the item-auction page. Query 1 is run first so that the number of bids made on the current auction can be counted if possible. However, if no bids have been made, this query will return nothing. In this case, query 2 is run to fetch the item and auction details without counting bids.

**fetch\_item\_categories :**

**SELECT c.categoryName FROM categories as c, itemCategories as ic**

**WHERE c.categoryID = ic.categoryID**

**AND ic.itemID = ?**

This query fetches all of the categories for a given item using its itemID.

**insert\_new\_bid :**

**INSERT INTO bids (userID, auctionID, bidAmount, bidDatetime) VALUES (?, ?, ?, ?)**

This query adds a new user bid to the bids table after it has been validated. The bid datetime is generated using PHP just before this function is called.

**get\_catagories :**

**SELECT categoryName FROM categories**

This query returns all of the categories in the database. This is used to populate the dropdown menu on the homepage and the add new listing page.

**fetch\_user\_email\_from\_username :**

**SELECT email FROM users WHERE username = ?**

This query returns a user’s current email address by using their username to locate their data.

**update\_email :**

**UPDATE users SET email=? WHERE username = ?**

This query is used to update a user’s email. The data is located using their username, which has a UNIQUE constraint.

**change\_password :**

**UPDATE users SET password = ? WHERE username = ?**

This query is used to update a user’s password. Their data is located using their username, which has a UNIQUE constraint. Hash functions are used earlier in the PHP code so that only the hashed version of the new password is entered into the database.

**fetch\_categoryid\_from\_category :**

**SELECT categoryID FROM categories WHERE categoryName = ?**

This query simply returns the category ID of a given category using its name. This is used to pass the category ID to functions below.

**insert\_item :**

**INSERT INTO items (itemName, sellerID, description) VALUES (?, ?, ?)**

**insert\_item\_category :**

**INSERT INTO itemCategories (itemID, categoryID) VALUES (?, ?)**

**insert\_auction :**

**INSERT INTO auctions (itemID, startPrice, reservePrice, startDatetime, endDatetime) VALUES ( ?, ?, ?, ?, ?)**

This set of queries is used to input the details for a new item and auction into the database. When a user adds a new listing, the insert\_item query first adds the item details into the items table. The insert\_item\_category query then links this to the chosen category by adding an entry into the itemCategories table using the ID of the item that was just added and a categoryID that is passed to it from the fetch\_categoryid\_from\_category function. The last query, insert\_auction, then just creates a new auction for this item and stores its details that the user has just provided.

**fetch\_search\_results :**

**Query 1**

**SELECT a.auctionID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime**

**FROM items as i, auctions as a**

**WHERE a.itemID = i.itemID**

**AND a.endDatetime > now()**

**AND itemName LIKE CONCAT('%',?,'%')**

**Query 1**

**SELECT a.auctionID, i.itemName, i.description, a.startPrice, a.reservePrice, a.startDatetime, a.endDatetime**

**FROM items as i, auctions as a, itemCategories as ic, categories as c**

**WHERE a.itemID = i.itemID**

**AND i.itemID = ic.itemID**

**AND ic.itemID = c.categoryID**

**AND a.endDatetime > now()**

**AND itemName LIKE CONCAT('%',?,'%')**

**AND c.categoryName = ?**

The two queries above are used by the search bar on the homepage to find items in the database using a text input with an optional category dropdown box. Query 1 is used when the user has not selected a category, so it searches for all auctions that contain an item with a name that contains the input substring. Query 2 is then used if the user did select a category, so it is the same as the first, but now restricts the search using the category name as well.

**close\_auctions:**

**SELECT a.auctionID FROM auctions as a WHERE a.endDatetime < now() AND a.endDatetime > addtime(now(), '-01:00')**

This query is part of a function that is run every hour to check and close auctions that have ended. The two functions below are called within this, but both require auctionIDs for each auction that has finished in the last hour to work, so the query above is used to retrieve this information.

**sendEmailToBidder :**

**Query 1**

**SELECT username, email, userID**

**FROM users**

**WHERE userID = (SELECT userID**

**FROM bids, auctions**

**WHERE auctions.auctionID = ?**

**AND bids.auctionID = auctions.auctionID**

**AND bids.bidAmount >= auctions.reservePrice**

**ORDER BY bidAmount DESC**

**LIMIT 1)**

**Query 2**

**INSERT into purchaseHistory (buyerID, auctionID) VALUES (?,?)**

**Query 3**

**INSERT INTO notifications (userID, auctionID, datetimeAdded) VALUES(?,?,NOW())**

This set of queries is part of a function that is run every hour to send out email notifications to the buyers of items when they have won an auction or is called when they have been outbid. Query 1 uses a sub-query to get the userID of the user who made the highest bid for a given auction and then this is used in the outer query to retrieve that user’s details. It returns nothing if the highest bid is under the reserve price, so this is the case where an auction ended with no bids over the reserve price and the function just returns. If user details were returned, a boolean input to the overall function then indicates whether an email needs to be sent to indicate that they are about to be replaced as the highest bidder or whether they have won an auction. If it is the situation where they have been outbid (i.e. the boolean is false), an email is sent and then Query 3 is used to add a notification for this user to the bids table. If it is the end of the auction and they are the highest bidder (i.e. boolean is true), the same process is used, but Query 2 is run before the email is sent to add the item to the winning user’s purchase history.

**sendEmailToSellerAtAuctionEnd :**

**Query 1**

**SELECT username, email, users.userID**

**FROM users, items, auctions, bids**

**WHERE auctions.auctionID = ?**

**AND auctions.itemID = items.itemID**

**AND items.sellerID = users.userID**

**AND auctions.auctionID = bids.auctionID**

**AND bids.bidAmount >= auctions.reservePrice**

**LIMIT 1**

**Query 2**

**SELECT username, email, users.userID**

**FROM users, items, auctions, bids**

**WHERE auctions.auctionID = ?**

**AND auctions.itemID = items.itemID**

**AND items.sellerID = users.userID**

**LIMIT 1**

**Query 3**

**INSERT INTO notifications (userID, auctionID, datetimeAdded) VALUES(?,?,NOW())**

This set of queries is part of a function that is run every hour to send out email notifications to the sellers of items when their auctions have finished. Query 1 is used to do two things: it is used to identify if any bids have been made on a given seller’s item that are over the reserve price and, if there are, it retrieves the seller’s details. Query 2 is used if the first query returns nothing, meaning that no bids over the reserve price were made on the item. In this case, it just retrieves the seller’s detail for the given auction so that an email notification can still be sent. This is done following the first two queries and then, once completed, the final query, Query 3, adds a notifications to the table for the given user and auction.

**fetch\_recommendations :**

**SELECT itemName, description, startPrice, reservePrice, startDatetime, endDatetime, dateOfRecommendation, auctionID**

**FROM recommendations, items, auctions**

**WHERE userID = ?**

**AND itemRecommendation = items.itemID**

**AND items.itemID = auctions.itemID**

**AND auctions.endDatetime > now()**

**ORDER by dateOfRecommendation DESC**

**LIMIT 5**

This query is used to fetch the latest 5 item recommendations for a given user from the recommendations table on the homepage when they login.

**gen\_reco\_item :**

**Query 1**

**SELECT itemcategories.categoryID , COUNT(bids.bidID) AS numberOfBids**

**FROM itemcategories, bids, auctions**

**WHERE auctions.auctionID = bids.auctionID**

**AND auctions.itemID = itemcategories.itemID**

**AND bids.userID = ?**

**GROUP BY itemcategories.CategoryID**

**ORDER BY numberOfBids DESC**

**Query 2**

**SELECT i.itemID, COUNT(b.bidID) as numberOfBids**

**FROM items as i, bids as b, auctions as a, itemCategories as ic**

**WHERE b.auctionID = a.auctionID**

**AND a.itemID = ic.itemID**

**AND i.itemID = ic.itemID**

**AND ic.categoryID = ?**

**AND a.endDatetime > now()**

**GROUP BY i.itemID**

**ORDER BY numberOfBids DESC**

**Query 3**

**INSERT INTO recommendations (userID, itemRecommendation, dateOfRecommendation) values (?, ?, ?)**

This set of queries is used to generate recommendations for the user based on the types of items that they have bid on the most and what other items of the same category are being bid on the most by other users. Query 1 is used to group the current user’s bids by item category and count the number that they have made for each one. Another function in the dbHelper class is then used to filter out the highest of these from the top of the results and pick one at random from this reduced array. Query 2 is then used to carry out the second part of the recommendations process, so each item that is still under auction with the same selected category is grouped by the total number of bids made on it. The same function as before is then used to select the item IDs with the highest number of bids and one is selected at random. This is the item that the system will recommend to the user when they next login, so Query 3 adds it to the recommendations table for that user. As this table has a UNIQUE constraint on the userID and itemID (itemRecommendation) pair, if the item is already in the table for the given user, this will fail, but a try-catch block simply exits from the function as normal without adding the item in this case.

**check\_watch\_item :**

**SELECT watchID FROM watchList WHERE userID = ? AND auctionID = ?**

This query checks to see if a given auction is already being watched by a given user. This is used on each item auction page to identify what type of button should be shown to the user, i.e. should they see the “Add to watchlist button” or the “Remove from watchlist button”?

**add\_watch\_item :**

**INSERT INTO watchList (userID, auctionID) VALUES (?, ?)**

This query inserts a given auction into the watchList table for a given user when they have indicated that they want to watch it using a button click on that particular auction page.

**remove\_watch\_item :**

**DELETE FROM watchlist WHERE userID = ? AND auctionID = ?**

This query deletes a given watched auction from the watchList table for a given user when they have indicated that they no longer want to watch it using a button click on that particular auction page.

References

[1] 1keydata.com. (2019). *Database Normalization*. [online] Available at: https://www.1keydata.com/database-normalization/ [Accessed 12 Mar. 2019].