Auction House

Group Project

Rasheed, Fatima

Summary

A bidding system was created where sellers can create auctions and buyers can browse them, bid, win auctions, receive emailed notifications and view product recommendations, following the design brief capabilities in the brief.

The report summarized the functionalities of the website and the design process of the bidding system we created. The design process follows the structure below:

- Creating an ER diagram
- Creating a database schema
- Creating database queries that would support the functionalities of the website

1. YouTube Video

A video was recorded to demonstrate how the bidding website works and that it follows the capabilities listed in the brief.

LINK: *https://www.youtube.com/watch?v=VM29olQksuw*

2. Entity Relationship Diagram

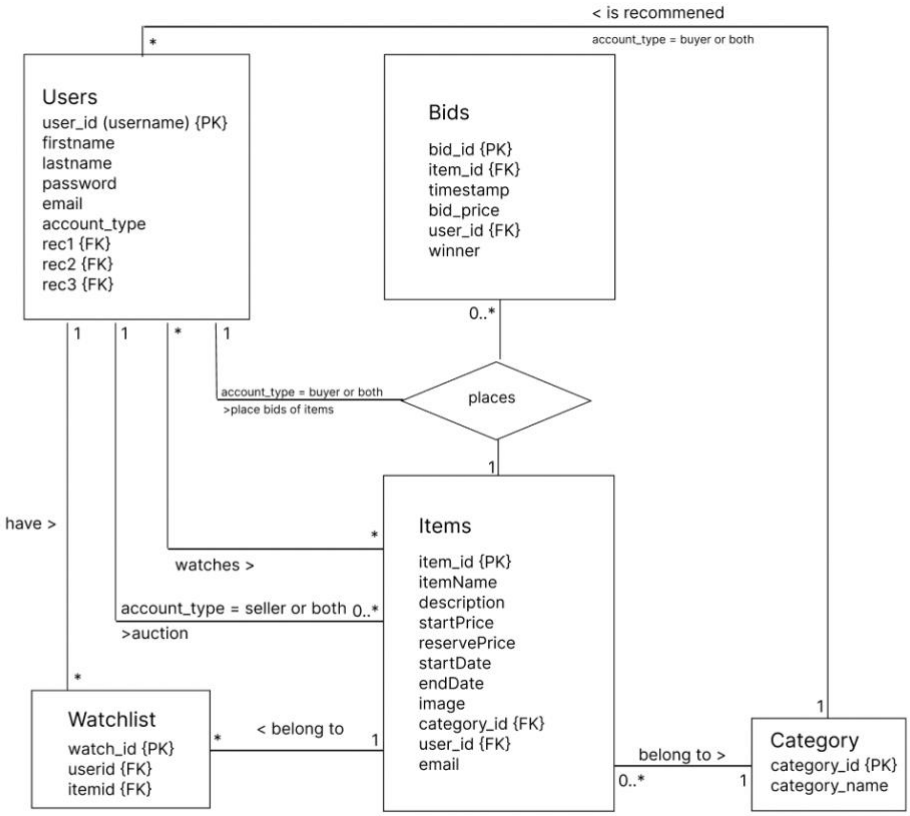

Figure 1 – ER Diagram

The user entity seen in our ER diagram encompasses different account types – buyers, sellers, or both – and these different types translate to different user privileges. Users registered as sellers or both can create auctions on items. An item is auctioned by a single seller (1: *). Additionally, all users can watch items (*:*). However, only users registered as buyers or both can place bids on items. A buyer can place any number of bids on a single item. This is represented by the higher order relationship.

To incorporate requirement 5 (as per the coursework brief), a Watchlist table was created, with "watch_id" as the primary key that uniquely represents an item a user is watching. Users can watch multiple items and an item can be watched by multiple users. This gives the relationships users-watchlist (1: *) and items-watchlist (1: *).

For requirement 6, there are the "rec1", "rec2" , and "rec3" attributes in the users table that act as foreign keys to the "Category" table. One category can be recommended to multiple users, but a single user can be recommended only one category per rec (one category for "rec1", one category for "rec2", one category for "rec3"). Users:category (*:1) This feature is only accessible by buyers.

3. Database Schema

Users (**user_id,** firstname, lastname, password, email, account_type, rec1, rec2, rec3)

Items (**item_id**, itemName, description, startPrice, reservePrice, startDate, endDate, image, category_id, user_id, email)

Category (**category_id**, category_name)

Bids (**bid_id**, item_id, timestamp, bid_price user_id, winner)

Watchlist (**watch_id**, userid, itemid)

The database schema above was created using a top-down method – by translating it from the ER diagram. The attributes in bold are the primary keys while the colored attributes are the foreign keys corresponding to the tables in the same color.

The parent (1)-child(*) relationships were translated like this: child gets a foreign key pointing to the parent key as an attribute. For example, items (child) got a foreign key ("user_id") pointing to the parent (users) when users (sellers) create auctions. The same logic follows for the relationship between category-items.

For *: * relationships, the standard practice is to create a dedicated table for the relationship. For capability 3, buyer browses items, this was not done because the process does not update or modify the data, it only reads it. However, for capability 5, where users are watching items, a watchlist table was created with the foreign keys pointing to the 2 tables. The higher order relationship user (seller) placed bids on items was represented by creating a dedicated table for the relationship ("bids"), which has its foreign key pointing to the users and items table.

The figure below represents the tables, attributes, and their type.

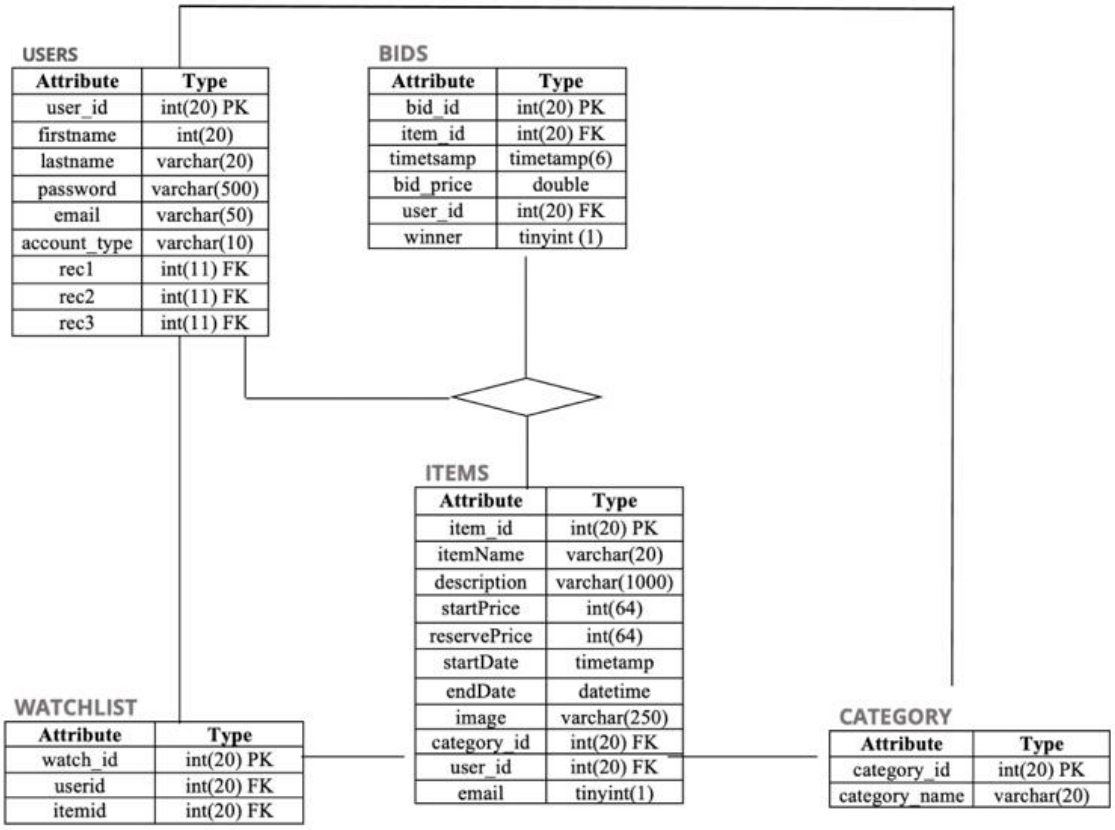

Figure 2 – Tables, and Attributes' Type

4. 3NF Analysis

For the database to be in 3rd Normal Form (3NF) it must have no transitive dependencies and be in 2nd Normal Form (2NF) which in turn demands it to be in 1st Normal Form (1NF) and have no partial dependencies. Hence to determine if the database is in 3NF one must determine that it is in 1NF and 2NF.

To be in 1NF three rules must be followed:

- Each cell must be occupied by a single value of a certain type.
- All values in each column must be of the same type, according to the column.
- Each column, in a table has a unique name.

As is evident from the ER diagram and the schema above the latter rule is followed since every table has a unique set of columns names. Certain entities are better coupled for intuitive understanding, such as a person's full name or the top three recommended categories in the "users" table. However, to follow the requirements of 1NF these have been split into mutually independent columns; "firstname", "surname", "rec1", "rec2" and "rec3". Finally, upon closer inspection into the values in the database, it is built in a way which makes it impossible to have values of different types be present in the same column. At first glance "email" column in "items" table could be interpreted as a storage place for strings containing email addresses, however that is not the case; it has a default value of one, and it is only influenced by the internal machinations of the web-auction's backend. Hence the value stored inside that column will always be 0 or 1. Upon closer examination it becomes clear that each table follows the 1NF requirements.

Moving on to 2NF evaluation, it is evident that the database follows 1NF. However, in addition to that it must also lack partial dependencies. These occur when a certain column depends on both the primary attribute and a non-primary attribute. For each table the non-primary attributes are entirely independent of each other. For instance, "user_id" in table "users" is the unique identifier for each corresponding attribute up to the "rec" attributes, due to the fact that those are input by the user upon registration, and recommendations are later determined by an algorithm according to the user's bids and collaborative filtering. Furthermore, there is no partial dependence in "bids" table since the bid is entirely dependent upon user input, with its properties independent of each other. One could argue that "bidprice" is dependent on the user's understanding of the value of the item, however that is not a solid relationship and each user's understanding of value is subjective. Finally, since this is a user-driven auction each item is unique, meaning each property is unique to that item; hence in the table "items" partial dependency is also impossible, in the form it is now. Due to the fact that "categories" table only contains two columns, partial dependency is impossible. The "watchlist" table has a single primary key given by "watch_id", so there are no partial dependencies (these only need to be checked for relationships with a composite primary key). This leads one to the conclusion that this database is 2NF normalized.

Finally, to satisfy the conditions of 3NF the database must have no transitive dependencies. This implies that no non-primary attribute in any table must be dependent on any other non-primary attribute, regardless of dependency on primary attributes. It would be tempting to include the "category_id" in the "bids" table as it would save an SQL query being executed within the collaborative trading algorithm, however this had to be excluded as it would violate the 3NF condition.

In conclusion, it is clear that database described in this report does satisfy the conditions to be in third normal form.

5. Listing and Explanation of Database Queries

Tables was created to show the queries used along with the explanation, following the order of the required capabilities for the website.

Process Registration

Query	Explanation
SELECT * FROM users WHERE email = '$email'	Checks for repeated emails to ensure that each user has a unique email
INSERT INTO users(firstname, lastname, password, email, account_type, rec1, rec2, rec3) VALUES (?,?,?,?,?,?,?,?)	Begins the insert process for a new user row in the users table, and the correct variables are sent into the insert function using the following line: mysqli_stmt_bind_param($stmt, "ssssssss", $firstname, $lastname, $password, $email, $accounttype, $interests[0], $interests[1], $interests[2]);
SELECT user_id FROM users WHERE email='$email'	Selects user id from the newly inserted row to set the user_id session variable

Check_user.php:

Query	Explanation
SELECT * FROM 'users' WHERE email='$email'	Check_user.php is called by password_email_functions.js which is in turn called when the user is registering. It asynchronously checks whether the email being entered is already in our database and displays this to the user

Login Result:

Query	Explanation
SELECT password FROM users WHERE email='$email'	Selects the password for the entered email to check if it matches the hashed password on the database and whether the email exists
SELECT user_id, account_type, firstname FROM users WHERE email='$email'	Selects the information we will need (eg buyer, seller, or both) and creates session variables

My Account:

Query	Explanation
SELECT * FROM `users` WHERE user_id = $user_id;	Selects the user's account information to display on their account page based on the $user_id variable that is determined from the session variables set

Create Auction:

Query	Explanation
SELECT * FROM category	Selects results that are displayed in category options when a user is filling out the create auction page

Create Auction Result:

Query	Explanation
SELECT MAX(item_id) FROM items	Used to find the current highest item_id so that we could assign the next item_id to a new item row
INSERT INTO items (item_id, itemName, description, startPrice, reservePrice, startDate, endDate, image, category_id, user_id, email) VALUES ('$id', '$title', '$details', '$startprice', '$reserveprice' , '$startdate', '$enddate','$new_img_name', '$categoryid','$userid',0)	Used when a new item auction is inserted into the items table if there are no errors in filling out the form

My listings:

Query	Explanation
SELECT * FROM items WHERE user_id=$userid	Pulls up the auctions that a seller has posted so they can easily access them on the My Listings page
SELECT MAX(bid_price) FROM bids WHERE item_id=$item_id	This statement fetches the highest current bid to display on the listings card

Browse page:

Query	Explanation
SELECT * FROM category	Selects all categories to populate the category option dropdown
SELECT * FROM items	This statement is called when nothing specified, or the user has selected all categories, and this allows the browse page to display all items
SELECT * FROM items WHERE enddate > CURRENT_TIMESTAMP	This statement is called when all categories are selected and the ticked auction ended checkbox

SELECT * FROM items WHERE itemName LIKE '%$keyword%'	This statement is called when all categories are selected, keywords have been specified keyword – shows all items where name matches keyword
SELECT * FROM items WHERE itemName LIKE '%$keyword%' and enddate > CURRENT_TIMESTAMP	This statement is called when all categories are selected, there are specified keyword, and there is a ticked auction ended checkbox – displays items where name matches keyword and enddate is later than current time
SELECT * FROM items,category WHERE items.category_id=category.category_id and category.category_id=$category	This statement is called when no keywords mentioned, but category is selected - so it displays items where category matches selection
SELECT * FROM items,category WHERE items.category_id=category.category_id and category.category_id=$category and enddate > CURRENT_TIMESTAMP	This statement is called when no keyword, but there is a category selected + ticked auction ended checkbox – it displays items where category matches selection and enddate is later than current time
SELECT * FROM items,category WHERE items.category_id=category.category_id and category.category_id=$category and itemName LIKE '%$keyword%'	This statement is called when there is a selected for keyword and category, so it displays items where name matches keyword and category matches selection
SELECT * FROM items,category WHERE items.category_id=category.category_id and category.category_id=$category and itemName LIKE '%$keyword%' and enddate > CURRENT_TIMESTAMP	This statement is called when there is a keyword and category + ticked auction ended checkbox – so it displays items where name matches keyword, category matches selection and enddate is later than current time
SELECT MAX(bid_price) FROM bids WHERE item_id=$item_id	This statement fetches the highest current bid to display on the listings card
ORDER BY startPrice ASC ORDER BY startPrice DESC ORDER BY endDate ASC	Used for the sort by button: Appended one of these 3 values to the end of the query depending on the selected value for the sort filter - order by price ascending or descending or order by enddate ascending

Listing page:

Query	Explanation
SELECT * FROM items WHERE item_id=$item_id	Grabs information about the selected item that will eventually be shown on the page
SELECT MAX(bid_price) FROM bids WHERE item_id=$item_id	Tracks the current highest bid for selected item which is also displayed on the listing page
SELECT * FROM `watchlist` WHERE userid=$user_id and itemid=$item_id"	This finds in the database whether the user is watching the item and displays the relevant information as a result.
SELECT * FROM `bids` WHERE item_id = $item_id and winner=1 and user_id=$user_id	This is line 178, it checks whether the current user has won this ended auction and then the document uses this info to display difference messages
SELECT bids.timestamp,bids.bid_price, users.firstname, users.lastname FROM bids,users WHERE bids.item_id=$item_id and bids.user_id=users.user_id ORDER BY bid_price DESC	Selects the information required for the bid history table

Place Bid:

Query	Explanation
SELECT * FROM items WHERE item_id=$itemid	Selects information about the item a user has placed a bid on. From this, we get info like startPrice and seller id to make sure that the bid is valid and not placed by the seller account
SELECT MAX(bid_price) FROM bids WHERE item_id=$itemid	Selects the highest bid_price on the item of interest, and only allows the new bid placed to pass if it is higher that the previous highest bid

SELECT MAX(bid_id) FROM bids	Selects the current highest bid id to increment it and set the id for the next bid placed
SELECT itemName FROM items WHERE item_id=$itemid	Selects the item name variable that we will pass later when watchlist and outbid emails are sent
SELECT user_id FROM bids where item_id=$itemid and bid_price=(SELECT MAX(bid_price) from bids where item_id=$itemid)	Checks if there are any bids placed on this item, and get the user_id information that will be needed when we send outbid email notifications
SELECT firstname, email FROM users WHERE user_id=$userid_lastbidder	Selects firstname and email data for the outbid emails if there are rows on the previous query statement
INSERT INTO bids (bid_id, item_id, timestamp, bid_price, user_id) VALUES ('$id','$itemid','$current_timestamp','$bid','$userid')	Inserts new bid row into the bids table taking account of id, item, timestamp, bid price, and user id of the user that placed the bid
SELECT * FROM watchlist WHERE itemid=$itemid	Selects all users who are watching the selected item so that we know which users want to be emailed about the new bid inserted
SELECT firstname, email FROM users WHERE user_id=$watchlistuser	Selects the variables we need to pass into the watchlist notification function

My bids:

Query	Explanation
SELECT DISTINCT item_id FROM bids WHERE user_id=$userid	Finds the item ids that the logged in user has bid on to display relevant bids

SELECT MAX(bid_price) FROM bids WHERE item_id=$item AND user_id=$userid	Find the users max bid for each item they have bid on to display on the listings
SELECT * FROM items WHERE item_id=$item	Used to select item information to display on listing card like img, title, description, and end time
SELECT MAX(bid_price) FROM bids WHERE item_id=$item	**This statement fetches the highest current bid** to display on the listings card

Purchases

Query	Explanation
SELECT `item_id`, `bid_price` FROM `bids` WHERE `user_id`=$userid and `winner`=1;	Selects the item and bid information from the bids table where the user id matches the id of the logged in user and where the bid was the winning bid. The winner variable is updated to 1 for the winning bid when the auction end date is passed.
SELECT * FROM `items` WHERE `item_id`=$item_id;	Selects information for the relevant purchased item to get the variables we need to pass into the print listing function

Watchlist_Funcs

Query	Explanation
INSERT INTO `watchlist`(`userid`, `itemid`) VALUES ($user_id,$item_id)	If a user has indicated that they want to watch an item, we insert their user id and the relevant item id into the watchlist table – this information is used to know who send emails to and what should be included in the email
DELETE FROM `watchlist` WHERE userid= $user_id and itemid = $item_id	If a user has indicated that they want to remove themselves from the watchlist for a certain item, we delete the relevant row from the watchlist table

Email page:

Query	Explanation
SELECT * FROM items WHERE endDate<CURRENT_TIMESTAMP AND email=0	Continuously checks for auctions that are ended where email notifications have not sent, if there are results we continue to check whether the auction was successful, and who receives the email notifications
SELECT * FROM users WHERE user_id=(SELECT user_id FROM items WHERE item_id=$itemid)	Used to fetch the seller name and email to pass as variables for the send to seller functions
SELECT * FROM bids WHERE item_id=$itemid	Checks whether there are any bids on the item, if there are none then we send the failed auction email to the seller
UPDATE items SET email=1 WHERE item_id=$itemid	We update the email boolean for each item when we have sent the necessary emails for it
SELECT MAX(bid_price) FROM bids WHERE item_id=$itemid	Selects the max bid price to send as a variable in the email functions to be display in the body text
SELECT * FROM users WHERE user_id=(SELECT user_id FROM bids WHERE item_id=$itemid and bid_price=$maxbid)	Used to select buyer email and firstname that are used to send as a variable in the email functions
SELECT reservePrice FROM items WHERE item_id=$itemid	Checks if there was a reservePrice, and if it isn't null then we check if max bid is higher than reservePrice

Recommendations

Query	Explanation
SELECT bid_id FROM bids WHERE user_id = $userid	Selects all bids made by the current user. It is used to check if the user made any bids. If they did: execute collaborative filtering algorithm and recommend items according to it. If they didn't: recommend items according to categories selected upon registration.
SELECT u.rec1, a.category_name crec1, u.rec2, b.category_name crec2, u.rec3, c.category_name crec3 FROM users u, category a, category b, category c WHERE user_id = $userid AND a.category_id=u.rec1 AND b.category_id=u.rec2 AND c.category_id=u.rec3	Used to make a table for top three recommendations for a particular user, with corresponding name and id. Later the ids are extracted to present items from corresponding categories and category names used to name the sections seen on the page.

SELECT * FROM items WHERE items.category_id=$recommends['rec1'] $sql.=" ORDER BY RAND() LIMIT 3	Selects random 3 items corresponding to first recommended category
SELECT * FROM items WHERE items.category_id=$recommends['rec2'] $sql.=" ORDER BY RAND() LIMIT 3	Selects random 3 items corresponding to second recommended category
SELECT * FROM items WHERE items.category_id=$recommends['rec3'] ORDER BY RAND() LIMIT 3	Selects random 3 items corresponding to third recommended category
SELECT b.bid_id, i.category_id, b.timestamp, b.bid_price, b.user_id FROM bids b, items i WHERE i.item_id = b.item_id	Collect all bids data and corresponding category_ids for each bid. Used by collaborative filtering algorithm to determine recommended categories
UPDATE users SET rec1=%s, rec2=%s, rec3=%s WHERE user_id = %s	Updates the recommended categories to those determined by collaborative filtering algorithm