Team 10 - Design Report

➢ **YouTube URL of video demonstration: https://youtu.be/m3lz3uyN1hQ**

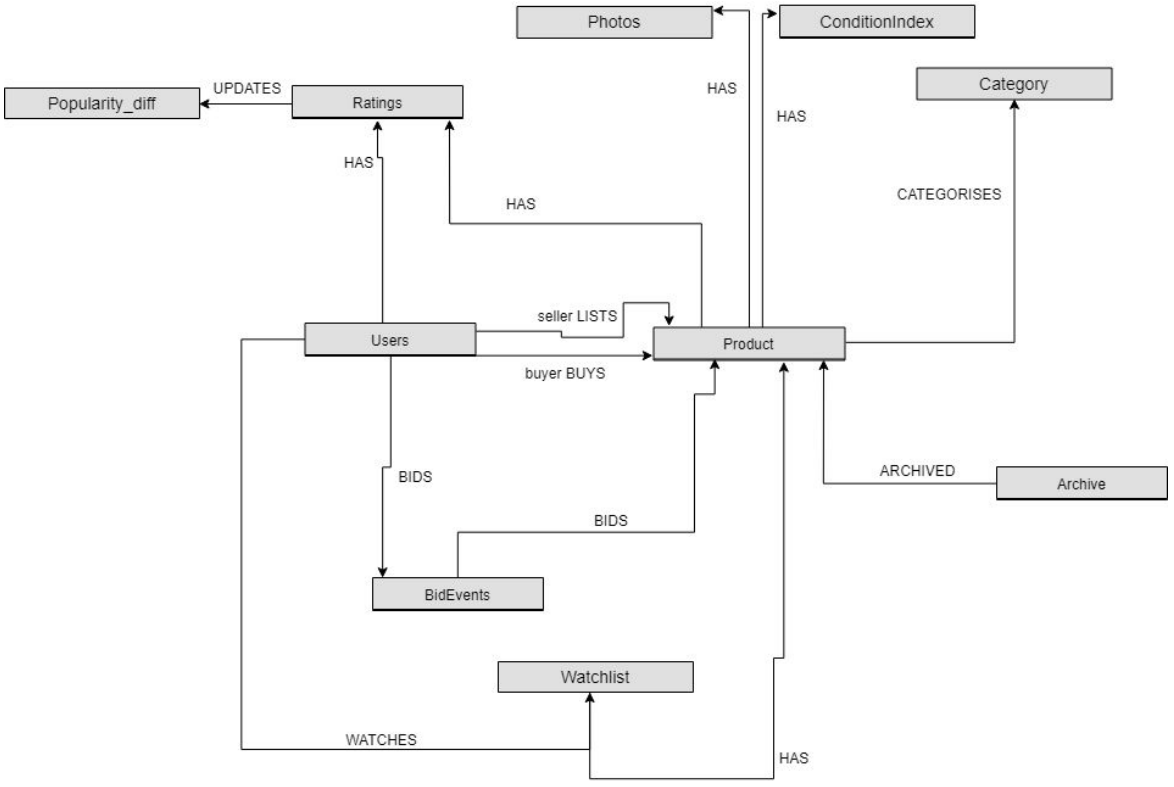
- Team 10's video demonstration can be viewed at the YouTube link above. The team demonstrates the functionality corresponding to each marking criteria, in order.

Criterion	**Capability**	**Video Timestamp**
1 (Users - Login)	User login function	0:02
1 (Users - Modify)	Modify user account details	0:14
1 (Users - Signout)	User signout function	0:40
1 (Users - Registration)	Register new user account	0:43
2 (Sellers - Auction Listing)	Create auction event and Listing of new items for sale	1:19
2 (Sellers - Auction)	Remove and Modify items	3:31
3 (Buyers - Search and Filter)	Search and filter auction event / keywords / category / condition for items to purchase	3:57
3 (Buyers - Refined Search)	Show all items and then filter with refined search criteria	4:25
4 (Buyers - Bid)	Demonstrate cannot bid on self-listed items	4:56
4 (Buyers - Bid)	Bids on others' auction items	5:15
4 (Buyers - Watch item) 5 (Buyers - Auction)	Buyer watchlists auction items	5:28
4 (Buyers - Compare)	Watch bids made by other users	6:00
5 (Buyers - Auction)	Email watchlist update (Spotify) and bid event update (iTunes) notifications	6:36
4 (Buyers - Award)	System award auction to winner at set endtime	6:44
4 (Buyers - Notify)	System notifies winner and seller of auction outcome via email	6:54
5 (Buyers - Non bidding item)	Buyer buys non-bidding item (i.e. not auctionable)	7:04

6 (Buyer - Recommendations)	Personalized search based on ratings and buying history	8:07
6 (Buyer - Recommendations)	Collaborative filtering	8:36

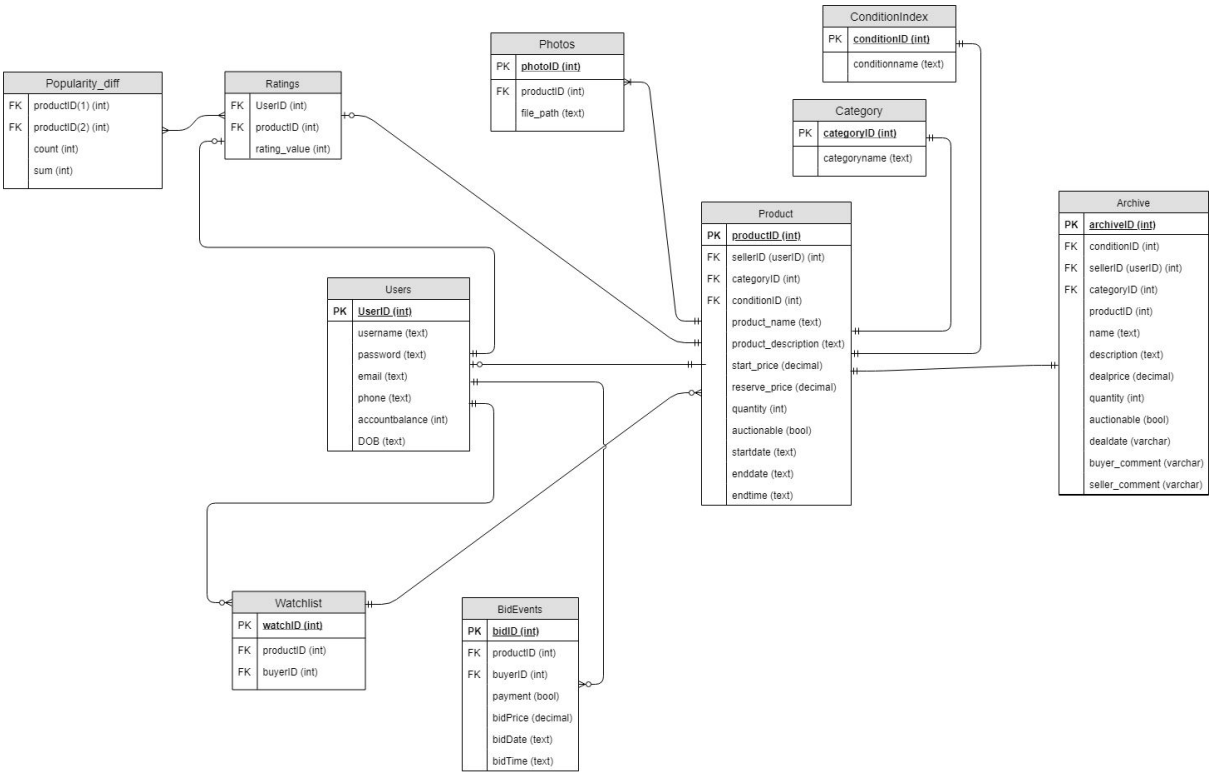
➢ **Entity Relationship Model**

The below model shows the key relationships between all the entities of the auction website.



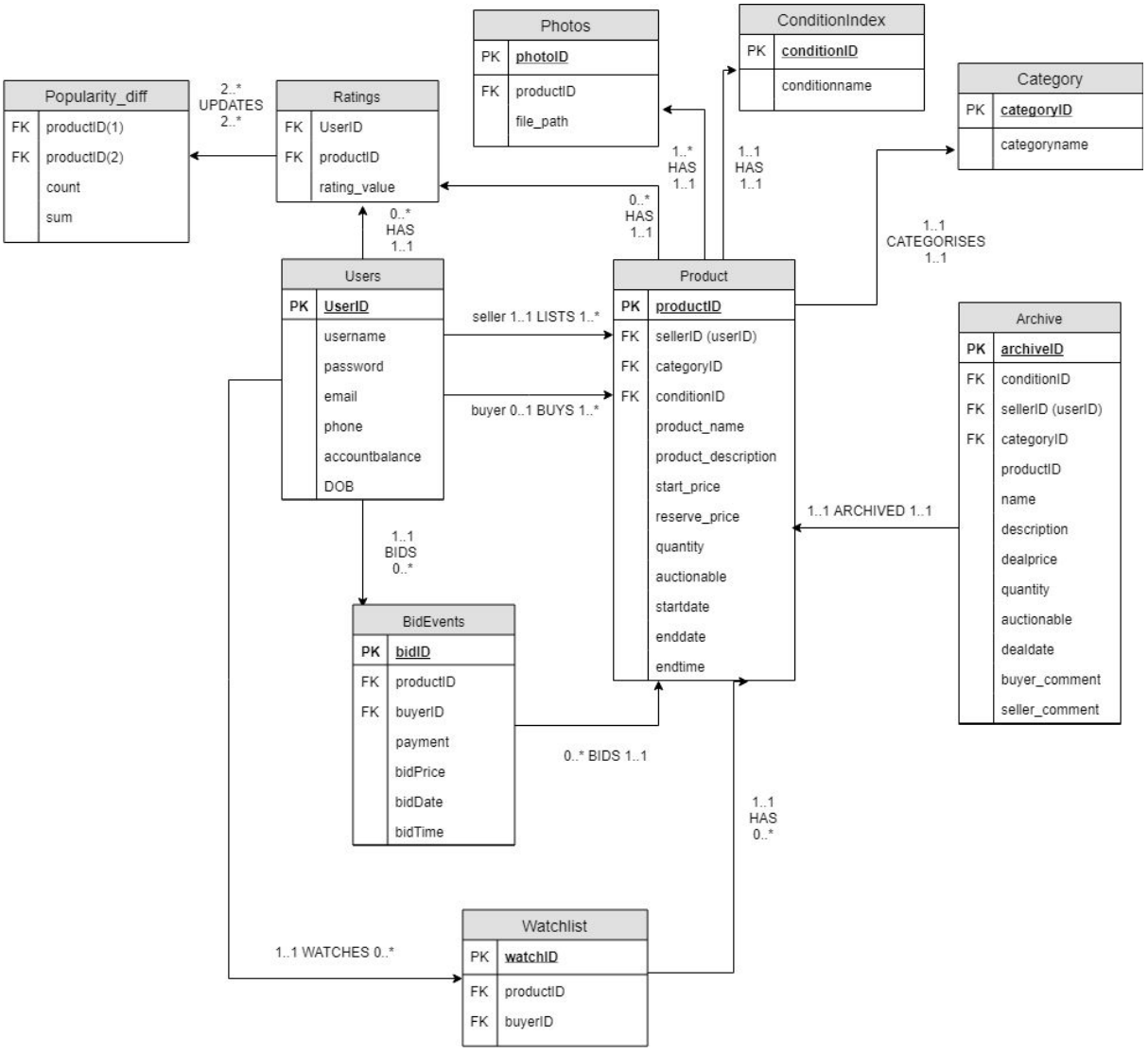
➢ **Database Schema**

This model shows the Primary Key and Foreign Key relationships between all the database tables, showing how exactly they connect. It also shows multiplicity notation, allowing us to determine any potential connection traps.



➢ **Entity Relationship Diagram**

This model effectively combines the previous two models, from which we ensure that Third Normal Form (3NF) is implemented.



Assumptions the ERD makes about the processes that use the database's data

The ERD assumes that SQL queries containing query arguments defined by the user's inputs, i.e. from a source "external to" the database, are in a correct and safe format. Therefore validation checking on the back-end is required for any instances where user inputs are used to interact with the database, to ensure no malicious or erroneous code is inserted into an SQL query.

All other processes use query arguments from validated sources "internal to" the database, and therefore cannot be subject to SQL injection or other unwarranted manipulation. These are outlined and discussed in the section below this section, entitled "User-input Independent Database Interactions".

User-input Dependent Database Interactions:

There are several website contexts where user input is used to define SQL queries:

1) User Login (User table)
2) New User Registration (User table)
3) My Profile - Modifying user account details (User table)
4) Seller Shop (Product table)
5) Seller Shop - Create New listing (Product table)
6) Buyer Bids (Bidevents table)
7) System archiving with user comments (Archive table)
8) Photo upload (Photos table)
9) Product rating upon successful bids during auction (Ratings table)

"User" table

UserID

- UserID is stored in a user's session variables once their login credentials are validated
- UserID is a critical identifier used throughout the SQL queries and is a unique integer number assigned to each user
- UserID is automatically created via Auto Increment for each user

Username

- Username format allows for alphanumeric characters and symbols, but does not allow for blank spaces
- Username is sanitised to prevent malicious user input
- Checking validation code exists to ensure that the username entered at new user account registration is **unique** i.e. doesn't currently exist within the database
- Username is also stored in a user's session variables to allow basic website customisation

Password

- Plaintext password is sent to server-side via POST method and is checked / validated against the stored hash password

Email

- Email format ensures that the '@' and email domain is included in user input
- Email is sanitised using the built-in PHP filter for email inputs

Phone Number

- Phone number is validated as a numeric format and of maximum length of 15 characters

Account Balance

- User input is sanitised and validated as numeric format
- Default account balance top up of £25 increments

DOB

- Date format is validated as YYYY-MM-DD
- DOB must be set such that a minimum age for a user account is 18 years old
- DOB cannot be in the future

"Product" table

Name

- Text input from the user.

Description

- Text input from the user.

"bidEvents" table

bidPrice

- User input is formatted through various checks such as whether it is higher than the latest bid price.
- User input is only allowed if the bidder is not the seller of the product.

"Archive" table

Buyer_comment

- Comments that can be left by the buyer.
- User input is of text form.

Seller_comment

- Comments that can be left by the seller.
- User input is of text form.

"Photos" table

photoID

- The uploaded file of the photoID is validated as an image by using a built-in PHP function getimagesize() which ensures that the file can rendered in HTML
- No other input (such as text input) is accepted.

“Ratings” table

rating_value

- User input is numeric, taking values between 1 to 10, with integer steps.
- The format is enforced by HTML tags such as min, max, step, type=”number”.

User-input Independent Database Interactions:

What follows is a list and explanation of contexts where the database tables which are not affected by external input: the interactions with and between these tables are internal and cannot be affected by a malicious user via a text box. They are for instance interactions that spawn from user interaction where the user just clicks a button without being able to attach any text to this interaction. After one of these interaction spawns, interactions that stem from it come into existence as a direct chain of internal database manipulation, again with no room for users to alter the interactions via text input or other means.

“Product” table

SellerID

- Set by the system when a user sells an item.

CategoryID

- Set by the system when a user sells an item.

ConditionID

- Set by the system when a user sells an item.

Reserve_price

- Set by the system when a user sells an item.
- Stored as decimal with two points instead of float for accuracy.
- Not seen by user, but is evaluated during auction ending.

Quantity

- User input through a drop down box, no formatting is required.

Auctionable

- Set by the system when a user sells an item.

Startdate

- User input when selecting a drop down list, hence no formatting needed.

Enddate

- Set by the system when a user sells an item.
- Will be extended by 30 days if auctionable product does not have bid events by the end of its auction.

Endtime

- User input when selecting a drop down list, hence no formatting needed.

“Archive” table

productID

- Set by the system when a bid is completed, product from Product will be moved into Archive.

Product_name

- Set by the system when a bid is completed.

Product_description

- Set by the system when a bid is completed.

Dealprice

- Set by the system when a bid is completed, the deal price is set as the bid price the auction winner pays.

Quantity

- Set by the system when a bid is completed.

categoryID

- Set by the system when a bid is completed.

conditionID

- Set by the system when a bid is completed.

buyerID

- Set by the system when a bid is completed, the buyerID is set as the winner of the auction.

sellerID

- Set by the system when a bid is completed.

auctionable

- Set by the system when a bid is completed

dealdate

- Set by the system when a bid is completed, the dealdate is set by the date of auction completion.

“Watchlist” table

watchID

- watchID is a primary key and is determined within the database using Auto Increment
- watchID cannot be amended by a user

productID

- productID is a foreign key and is determined within the database using Auto Increment within the Product table
- productID cannot be amended by a user

buyerID

- buyerID is a foreign key and is determined within the database using Auto Increment within the Users table
- buyerID cannot be amended by a user

“Category” table

categoryID

- categoryID is defined by the database administrator at initialization
- categoryID cannot be amended by a user

categoryname

- categoryname is defined by the database administrator at initialization
- categoryname cannot be amended by a user

“ConditionIndex” table

conditionID

- conditionID is defined by the database administrator at initialization
- conditionID cannot be amended by a user

conditionname

- conditionname is defined by the database administrator at initialization
- conditionname cannot be amended by a user

“bidEvents” table

productID

- productID is set by the system when the bid process succeeds.

buyerID

- buyerID is set by the system when the bid process succeeds.

payment

- Payment is set during auction checking by the system.

bidDate

- bidDate is set by the system when the bid process succeeds.

“Ratings” table

userID

- userID is set by the system when the bid process succeeds.

productID

- productID is set by the system when the bid process succeeds.

“Popularity_diff” table

productID1

- productID1 is set by the system when the bid process, and rating process succeeds.

productID2

- productID2 is set by the system when the bid process, and rating process succeeds.

count

- count is set by the system when the bid process, and rating process succeeds.

sum

- sum is set by the system when the bid process, and rating process succeeds.

➢ **Listing of database schema (explanation of translation into the ERD)**

The referential integrity constraints are outlined for each table within the database and are presented below. Within each table, the primary key and foreign key relationships are highlighted, along with the respective Create Read Update Delete (“CRUD”) rights associated for each foreign key.

Table Name	**Primary key**	**Alternate key(s)**	**Foreign key(s)**
users	userID	N/A	N/A
bidEvents	bidID	N/A	FOREIGN KEY (buyerID) REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE, FOREIGN KEY (productID) REFERENCES Product(productID) ON UPDATE CASCADE ON DELETE CASCADE
product	productID	N/A	FOREIGN KEY (sellerID) REFERENCES Users(userID) ON UPDATE CASCADE, FOREIGN KEY (categoryID) REFERENCES Category(categoryID) ON UPDATE CASCADE, FOREIGN KEY (conditionID) REFERENCES Conditionindex(conditionID) ON UPDATE CASCADE
archive	archiveID	N/A	FOREIGN KEY (sellerID) REFERENCES Users(userID) ON UPDATE CASCADE, FOREIGN KEY (categoryID) REFERENCES Category(categoryID) ON UPDATE CASCADE, FOREIGN KEY (conditionID) REFERENCES Conditionindex(conditionID) ON UPDATE CASCADE
watchlist	watchID	N/A	FOREIGN KEY (productID) REFERENCES Product(productID) ON UPDATE CASCADE ON DELETE CASCADE, FOREIGN KEY (buyerID) REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE
popularity_diff	Composite Key {productID(1), productID(2)}	N/A	FOREIGN KEY productID(1) REFERENCES Product(productID), FOREIGN KEY productID(2) REFERENCES Product(productID)
conditionIn	conditionID	N/A	N/A

dex			
category	categoryID	N/A	N/A
photos	photoID	N/A	FOREIGN KEY (productID) REFERENCES Product(productID) ON UPDATE CASCADE ON DELETE CASCADE
ratings	Composite Key {UserID, productID}	N/A	FOREIGN KEY (UserID) REFERENCES Users(UserID), FOREIGN KEY (productID) REFERENCES Product(productID)

➢ Analysis showing that the database schema is in its Third Normal Form (3NF)

First Normal Form (1NF)
Our database schema satisfies First Normal Form because each set of related data is contained within a separate table, and each of these separate tables are identified by a Primary Key. The intersection of each row and column in the tables in our database schema, in other words, contain one and only one value each. There are no “merged” cells, in effect. Every cell has at least one, and at most one, data entry within it. This creates significant data redundancy, which could be subject to update anomalies. Therefore it is important to progress the database schema to 2NF.

As an example, if a user puts in 4 bid events (makes four bids) on the same product, the database schema will not “group” or “merge” them under that user and that product, but will create 4 separate records within the “bidevents” table, representing each bid made.

Second Normal Form (2NF)
We could have accomplished full 2NF but after deliberation and consultation with an Academic Teaching Assistant, we decided to relax this for the Users table. This is because it is far easier to query an auto-incrementing integer “userID” Primary Key than it is to query a character-string “username” Primary Key. Thus we relax the partial dependency restriction and allow a partial dependency between “username” and “userID” in the Users table.

Except for this, our database schema is in Second Normal Form because it satisfies First Normal Form and there are no (other) partial dependencies (where a non-prime attribute depends on a proper subset of the Primary Key) within the tables. Any partially dependent relations have been moved into separate tables that are linked via Foreign-Key relations. In this way we evade update anomalies (including insertion, deletion, and modification anomalies).

Third Normal Form (3NF)
Our database schema satisfies Third Normal Form because it satisfies Second Normal Form and there are no transitive dependencies (where a non-prime attribute depends on another non-prime attribute) within the tables.

As an example, the two tables Users and Product could have originally been one table (e.g. a Product table with productID, sellerID, sellerName, productName, where productID would be the Primary Key, and where there would be a transitive dependency between the non-prime attributes sellerName and sellerID). We have however created two separate tables, Users and Product, where no non-prime attribute is dependent on any other non-prime attribute.

➢ List and explanation of Database queries (preliminary notes)

- Validation against SQL injection / malicious inputs

Our websites processes a large range of user inputs in multiple contexts, from user registration, user sign in, product search and filtering, item listings of auctionable items, etc. Where user input is accepted via HTML forms on the website, we implemented front-end and back-end validation checks to prevent inaccurate or unsafe user input.

Front-end validation on input forms is achieved using specific HTML5 formatting attributes in the HTML tags, such as regular expressions defined for “pattern” and applying “type” (e.g. date, number, text) to input tags to restrict input values (i.e. a defined maximum and minimum). JavaScript is also used to perform basic input checking within the webpage. Any malicious inputs are detected before form submission, with error messages displayed next to input field to facilitate correct user inputs.

Only correct and validated data can be submitted. The POST method, instead of GET, is used across our website to submit the forms to the server, to account for the large volume of submitted data. The POST method also prevent users from viewing the variables and values submitted via forms.

To cater for situations in which HTML5 formatting is not supported by certain browsers (i.e. older versions of Internet Explorer or Firefox) or where Javascript has been disabled by users, **back-end validation** is also performed using PHP after the data is submitted to the server. User inputs were sanitised before querying the database, including checking for empty values (*empty()*), searching for sensitive mysql strings (*preg_match()*) in text, trimming whitespace at the beginning and end of the user input (*trim()*), checking for correct data types (*is_a()*,*is_numeric()*, *date_create_from_format()*) and escaping characters (*mysqli_escape_from_string()*). In the case of invalid inputs, error messages are returned to the user within the webpage and no SQL query is executed.

For critical account details such as email address and password, the back-end validation ensured that repeated matching inputs were provided by the user. Passwords are stored in hashed form as a 60-character string using *password_hash()* PHP function and the *password_BCRYPT* algorithm.

The database queries were performed using procedural form prepared statements to ensure that SQL injection attacks or malicious inputs were prevented. The user-defined query parameters are passed to the SQL query as keyword arguments and the data types of the arguments are explicitly specified through *mysqli_stmt_bind_param()*. This provides for an explicit and intentional check on the user-input and prevents the execution of malicious SQL code.

- **Connection to Database and Queries**

Standardised scripts were written for initiating database connections and were reused within multiple PHP scripts via PHP *include* statements, maximizing code efficiency. The header and footer navbar functionality was also achieved using PHP *include* statements on the relevant pages of the website.

The tables contained within our database were organised such that most of the SQL queries are made upon single tables, without requiring joins between multiple tables. This enables more efficient access to the database, allowing for lower search and access times if the database was to be scaled to tens of thousands of products. In some SQL queries relating to the Buyer's Purchase History, inner joins are used between "Archive" table and "Users" table.

- **Recommendation System (Collaborative Filtering)**

Collaborative filtering is a method of making personalised recommendations based on a database of user preferences. The rating-based collaborative filtering[1], is used to build our recommendation system.

The first table - 'Ratings' holds the ratings for each user and their rated product. Another table - "Probability_Diff" holds the average rating difference between every product, as well as the number of times an item is bidded on. The system predicts the rating a user will give to a unrated item by using these two tables.

Besides the personalised recommendation, a non-personalised one can be obtained by not considering the user when making a prediction. The items that will be recommended are the ones with the highest positive average rating difference.

[1] Daniel Lemire. *Implementing a Rating-Based Item-to-Item Recommender System in PHP/SQL.* Research Gate. Jan 2005.

- **Image Storage**

Image storing is done through using Photos table by storing file paths, and having an "./uploads" folder to store images. This is chosen for scalability as it is easier to manage text rather than file formats.

➢ **<u>List and explanation of Database queries (code and explanation)</u>**

SQL Query	**Action / Purpose**	**Criterion / Capability**

$host = $_SERVER['HTTP_HOST'];
 	
if (strpos($host,"localhost")===0){
  $connection = mysqli_connect("localhost", "root", "", "ebaydb");
  if (mysqli_connect_errno()){
  echo 'Failed to connect to the MySQL server: '. mysqli_connect_error();}
	Database connection to localhost	ALL

$connection->close();	All SQL query ends with this line to close connection to database	ALL

$sql_users = "SELECT username FROM users WHERE username = ?";
$result_users = mysqli_prepare($connection, $sql_users);
mysqli_stmt_bind_param($result_users, 's', $username);
mysqli_stmt_execute($result_users);
$result = mysqli_stmt_get_result($result_users);
$rows_users = ($result->num_rows);
	Users to login with valid username, as compared to username stored in database	1
		(Users)
		loginpage.php

// Username found in users table
if ($rows_users == 1) {
  // Check if password matches in users table
  $sql_users_pw = "SELECT * FROM users WHERE username = ?";
  $result_users_pw = mysqli_prepare($connection, $sql_users_pw);
 	
mysqli_stmt_bind_param($result_users_pw, 's', $username);
 	
mysqli_stmt_execute($result_users_pw);
  $result = mysqli_stmt_get_result($result_users_pw);
  while($row = $result->fetch_assoc()) {
  if (password_verify($user_pass, $row["password1"])) {
  echo "Successfully logged in"."
";
  $userID = $row["userID"];
	Users to login with valid password, as compared to hashed password stored in database	1
		(Users)
		loginpage.php

createSession($userID, $username);
}
}
}		

// Check for duplicate username in users table
$sql_users = "SELECT username FROM users WHERE username = ?";
$result_users = mysqli_prepare($connection, $sql_users);
mysqli_stmt_bind_param($result_users, 's', $username);
mysqli_stmt_execute($result_users);
$result = mysqli_stmt_get_result($result_users);
$rows_users = ($result->num_rows);
echo "connected to users" . "
";
 	
if ($rows_users == 0) {
// Username is available
}
	Check for potential duplicate username when new users register a new user account to database	1
		(Users)
		regNewUser.php

// INSERT new row to database
$hash = password_hash($password, PASSWORD_BCRYPT);
$sql_insert = "INSERT INTO users (username, password1, email, phone, accountbalance, DOB) VALUES (?, ?, ?, ?, ?, ?)";
$insert_user = mysqli_prepare($connection, $sql_insert);
$zero = 0;
mysqli_stmt_bind_param($insert_user, 'ssssis', $username, $hash, $email, $phoneNo, $zero, $DOB_str);
$result = mysqli_stmt_execute($insert_user);
	Add new user account to the users table as a new row	1
		(Users)
		regNewUser.php

$sql="SELECT * FROM Category";
$result=$connection->query($sql);
 	
if ($result->num_rows>0){
while($row=$result->fetch_assoc()){
$_SESSION["category_all"][$row["categoryID"]]=$row["categoryname"];
}
} else {
	Obtain the latest category ID/name conversion from Category Table.	2
		(Sellers)
		product.php

echo "Error: ". $sql . " " . $connection->error; }		

$sql="SELECT * FROM ConditionIndex";
$result=$connection->query($sql);
 	
if ($result->num_rows>0){
  while($row=$result->fetch_assoc()){
$_SESSION["condition_all"][$row["conditionID"]]=$row["conditionname"];
  }
} else {
echo "Error: ". $sql . "
" .
$connection->error;
}
	Obtain the latest condition ID/name conversion from ConditionIndexTable.	2
		(Sellers)
		product.php

$product_name=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["product_name"]); $productID=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["productID"]); $product_description=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["product_description"]); $start_price=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["start_price"]); $reserve_price=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["reserve_price"]); $quantity=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["quantity"]); $conditionname=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["conditionname"]); $categoryname=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["categoryname"]); $sellerID=mysqli_real_escape_string($	Remove sensitive mysql characters from validated user inputs and insert a new item into Product Table.	2 (Sellers) **product.php**

```
connection,$_SESSION["editlisting"]["sellerID"]);

$auctionable=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["auctionable"]);

$startdate=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["startdate"]);

$enddate=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["enddate"]);

$endtime=mysqli_real_escape_string($connection,$_SESSION["editlisting"]["endtime"]);

$photos_file_path=$_SESSION["editlisting"]["photos"]["file_path"];

 $sql="INSERT INTO Product (product_name,product_description,start_price,reserve_price,quantity,categoryID,conditionID,sellerID,auctionable,startdate,enddate,endtime) VALUES('$product_name','$product_description','$start_price','$reserve_price','$quantity','$categoryID','$conditionID','$sellerID','$auctionable','$startdate','$enddate','$endtime')";

if ($connection->query($sql)==TRUE){
echo "New record successfully created for product";
} else {
echo "Error: ". $sql . "<br>" . $connection->error;
}

//fetch new productID
$sql="SELECT LAST_INSERT_ID()";
$result=$connection->query($sql);
if ($result->num_rows>0) {
    while($row = $result->fetch_assoc()){
        $productID = $row["LAST_INSERT_ID()"];
```

}
} else {
echo("Error: " . $sql . "
" . $connection->error);
}
 	
// upload photo to database
foreach ($photos_file_path as $file_index=>$val) {
$photo_arr['productID'] = $productID;
$photo_arr['photoID'] = 0;
$photo_arr['file_path'] = mysqli_real_escape_string($connection, $val);
$new_photo_arr= set_photo($photo_arr, "insert");
$_SESSION["editlisting"]["photos"]["photoID"][$file_index] = $new_photo_arr['photoID']; // update photos with new photoID
}		

$sql="UPDATE Product
SET product_name='$product_name',
 	
product_description='$product_description',
start_price='$start_price',
reserve_price='$reserve_price',
quantity='$quantity',
categoryID='$categoryID',
conditionID='$conditionID',
auctionable='$auctionable',
enddate='$enddate',
endtime='$endtime'
WHERE productID=$productID";
if ($connection->query($sql) === TRUE) {
echo "Record updated successfully for Product table";
} else {
echo "Error updating record: " . $connection->error;
}
	Update the item information in the product table	2
		(Sellers)
		checktime.php

$sql="DELETE FROM Product WHERE productID='$productID'"; $result=$connection->query($sql); if ($result==TRUE){	Delete the item from Product Table and echo “.. Successfully” if successful.	2 (Sellers) **removelisting.php**

echo "Record deleted successfully";
} else {
echo "Error: ". $sql . "
" .
$connection->error;
}		

// FIND ALL RELEVANT LISTINGS
$s_all_active_listings = []; //
array has pID as its keys to identify
unique items that has been searched
 	
foreach
($search_unique_words as $word) {
// echo "
word: ".$word;
 	
$_SESSION["product_search_criteria"]
=["keyword",$word];
include 'fetchactivelisting.php';
 	
foreach
($_SESSION["all_active_listings"] as
$listing) {
$pID = $listing["productID"];
 	
if (array_key_exists($pID,
$s_all_active_listings)) {
 	
$s_all_active_listings[$pID]["search_co
unt"] += 1;
} else {
$listing["search_count"]
= 1; // initialise number of times
searched (the higher the better for our
search)
 	
$s_all_active_listings[$pID] = $listing;
// force unique products only
}
}
}
 	
// SORT LISTING BY
SEARCH_COUNT DESCENDING (to
show the most relevant results first)
$s_all_active_listings =
array_values($s_all_active_listings);
 	
$_SESSION['all_active_listings'] = [];
 	
foreach ($s_all_active_listings
	Get all relevant listings related to the words typed in search bar.	3
		(Search)
		buyershop.php

as $key=>$val) {
$key_count[$key] = $val["search_count"];
}
arsort($key_count); // sort by highest count
 	
// RENAME KEYS TO 0, 1, 2, 3 ...
foreach ($key_count as $key=>$val) {
 	
array_push($_SESSION['all_active_listings'], $s_all_active_listings[$key]);
}		

$checked=in_array($_POST["categorylist"],array("Category","Electronics","Food","Fashion","Home","Health & Beauty","Sports","Toys & Games","Art & Music","Miscellaneous"));
 	
$checked=in_array($_POST["conditionlist"],array("Condition","New","Refurbished","Used / Worn"));
 	
if ($checked){
 	
if ($_POST["categorylist"]!="Category") && ($_POST["conditionlist"]!="Condition")) {
 	
//search based on category and condition
 	
$category=array_search($_POST["categorylist"],$_SESSION["category_all"]);
 	
$condition=array_search($_POST["conditionlist"],$_SESSION["condition_all"]);
echo $condition;
 	
$_SESSION["product_search_criteria"]=["c&c",$category,$condition];
	Search based on category and/or condition of items	3
		(Search)
		buyershop.php

```
                include 'fetchactivelisting.php';

        }elseif (($_POST["categorylist"]!="Category") && ($_POST["conditionlist"]=="Condition") ){

                //search based on category

$category=array_search($_POST["categorylist"],$_SESSION["category_all"]);

$_SESSION["product_search_criteria"]=["category",$category];

                include 'fetchactivelisting.php';

        }elseif (($_POST["categorylist"]=="Category") && ($_POST["conditionlist"]!="Condition")) {

                //search based on condition

$condition=array_search($_POST["conditionlist"],$_SESSION["condition_all"]);

$_SESSION["product_search_criteria"]=["condition",$condition];

                include 'fetchactivelisting.php';

        } else {

$_SESSION["product_search_criteria"]=["all",""];
                include 'fetchactivelisting.php';
        }
```

} 		
$sql = "INSERT INTO bidEvents (productID, buyerID, payment, bidPrice, bidDate, bidTime) VALUES ('$productID', '$buyerID', '$payment', '$bidPrice', '$date_today', '$time_today')"; $result = $connection->query($sql);	Insert details into bidEvents	4 (Auction) **bid_product_interface.php**

$bids = [];
if ($condition == "latest") {
$sql = "SELECT * FROM bidEvents WHERE bidPrice=(
SELECT MAX(bidPrice) FROM bidEvents WHERE productID = '$productID'
);";
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row=$result->fetch_assoc()) {
$bid_arr['bidID'] = $row['bidID'];
$bid_arr['productID'] = $row['productID'];
$bid_arr['buyerID'] = $row['buyerID'];
$bid_arr['payment'] = $row['payment'];
$bid_arr['bidPrice'] = $row['bidPrice'];
 	
array_push($bids, $bid_arr);
}
 	
// echo("Received bid event successfully.");
} else {
$bid_arr['bidID'] = 0;
$bid_arr['productID'] = 0;
$bid_arr['buyerID'] = 0;
$bid_arr['payment'] = 0;
$bid_arr['bidPrice'] = 0;
array_push($bids, $bid_arr);
	Get bidEvents	4
		(Auction)
		bid_product_interface.php

}
 	
} elseif ($condition == "all") {
$sql = "SELECT * FROM bidEvents WHERE productID = '$productID'
ORDER BY bidID DESC";
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row=$result->fetch_assoc()) {
$bid_arr['bidID'] = $row['bidID'];
$bid_arr['productID'] = $row['productID'];
$bid_arr['buyerID'] = $row['buyerID'];
$bid_arr['payment'] = $row['payment'];
$bid_arr['bidPrice'] = $row['bidPrice'];
$bid_arr['bidDate'] = $row['bidDate'];
$bid_arr['bidTime'] = $row['bidTime'];
array_push($bids, $bid_arr);
}
echo("Received bid event successfully.");
} else {
$bid_arr['bidID'] = 0;
$bid_arr['productID'] = 0;
$bid_arr['buyerID'] = 0;
$bid_arr['payment'] = 0;
$bid_arr['bidPrice'] = 0;
$bid_arr['bidDate'] = 0;
$bid_arr['bidTime'] = 0;
array_push($bids, $bid_arr);
}
}		

$sql = "INSERT INTO watchlist (productID, buyerID) VALUES ('$productID', '$buyerID')"; if ($connection->query($sql)==TRUE) { echo("Inserted new watchlist item.\n");	Add product to user's watchlist	5 (Watch) **watch_product _ interface.php**

} else {
  echo("Error: " . $sql . "
" . $connection->error);
}		

$sql="DELETE FROM watchlist WHERE buyerID = '$buyerID' AND productID = '$productID'";
if ($connection->query($sql)==TRUE) {
  echo("Deleted watch event successfully.");
} else {
  echo("Error: " . $sql . "
" . $connection->error);
}
	Remove product from user's watchlist	5
		(Watch)
		watch_product _ interface.php

$watches = [];
$sql = "SELECT * FROM Watchlist WHERE buyerID = '$buyerID'
  ORDER BY watchID DESC";
$result = $connection->query($sql);
if ($result->num_rows>0) {
  while ($row=$result->fetch_assoc()) {
  $watch_arr['watchID'] = $row['watchID'];
  $productID = $row['productID'];
  $watch_arr['buyerID'] = $row['buyerID'];
  // get product name from product id and product table
  $sql = "SELECT * FROM Product WHERE productID = '$productID'";
  $result1 = $connection->query($sql);
  while ($row=$result1->fetch_assoc()) {
  $watch_arr['productName'] = $row['product_name'];
  $watch_arr['endDate'] = $row['enddate'];
  $watch_arr['endTime'] = $row['endtime'];
  // get seller name from seller id and user table
  $sellerID = $row['sellerID'];
  $sql = "SELECT * FROM Users WHERE userID = '$sellerID'";
  $result2 = $connection->query($sql);
  while
	Display to user their watchlist (showing product name, seller name, and auction end date and end time)	5
		(Watch)
		watchtable_interface.php

($row=$result2->fetch_assoc()) {
$watch_arr['sellerName'] = $row['username'];
}
}
array_push($watches, $watch_arr);
}		

// get username of user who placed the bid
$sql="SELECT * FROM Users WHERE userID = '$bidderID'";
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row=$result->fetch_assoc()) {
$latestBidderName = $row['username'];
}
}
// get name of product being bid on / watched
$sql="SELECT * FROM Product WHERE productID = '$productID'";
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row=$result->fetch_assoc()) {
$productName = $row['product_name'];
}
}
// for each user who is watching this product:
$email_to_arr = [];
$is_watching = "yes";
// get names and emails of all users watching this product
$sql="SELECT * FROM Watchlist WHERE productID = '$productID'";
$result1 = $connection->query($sql);
if ($result1->num_rows>0) {
while ($row=$result1->fetch_assoc()) {
$watcherID = $row['buyerID'];
$sql = "SELECT * FROM Users WHERE userID = '$watcherID'";
$result2 = $connection->query($sql);
	On new bid or end auction, send email update to all users watching the relevant product (and all users who have placed a bid on the relevant product)	5
		(Watch)
		update_watching.php

```
        if ($result2){
            while
($row=$result2->fetch_assoc()) {

$email_to_arr['watcherName'] =
$row['username'];

$email_to_arr['watcherEmail'] =
$row['email'];
            }
        }
    }
} else {
    $is_watching = "no";
}

if ($is_watching == "yes") {
    foreach ($email_to_arr as
$email_to=>$val) {
        // want to tell users that User has
made BidPrice on Product
        $subject = "New bid on
".$productName;
        $body = "Hey
".$email_to_arr['watcherName']."!\nUse
r ".$latestBidderName." has made a
new bid of ".$bidPrice." on ".$productN
        $altbody = "Someone has made a
bid on a product that you are
watching!";
        $watchingUserEmail =
$email_to_arr['watcherEmail'];
        $emailee_name =
$email_to_arr['watcherName'];

send_to_email($watchingUserEmail,
$subject, $body, $altbody,
$emailee_name);
    }
    // for each user who has made a bid
on this product:
    // get names and emails of all users
who have made a bid on this product
    $sql="SELECT * FROM BidEvents
WHERE productID = '$productID'";
    $result3 = $connection->query($sql);
    if ($result3->num_rows>0) {
        while
($row=$result3->fetch_assoc()) {
            $bidderID = $row['buyerID'];
```

```
            $sql = "SELECT * FROM Users
WHERE userID = '$bidderID'";
            $result4 =
$connection->query($sql);
            if($result4){
                while
($row=$result4->fetch_assoc()) {

$email_to_arr['bidderName'] =
$row['username'];

$email_to_arr['bidderEmail'] =
$row['email'];
                }
            }
        }
    }
} else {
    $is_watching = "no";
}

if ($is_watching == "yes") {
    foreach ($email_to_arr as
$email_to=>$val) {
        // want to tell users that User has
made BidPrice on Product
        $subject = "New bid on
".$productName;
        $body = "Hey
".$email_to_arr['watcherName']."!\nUse
r ".$latestBidderName." has made a
new bid of ".$bidPrice." on
".$productName."!\nYou are receiving
this because you have an active bid on
this product.\nHave a nice day!\nFake
ebay";
        $altbody = "Someone has made a
bid on a product that you are
watching!";
        $watchingUserEmail =
$email_to_arr['watcherEmail'];
        $emailee_name =
$email_to_arr['watcherName'];

send_to_email($watchingUserEmail,
$subject, $body, $altbody,
$emailee_name);
    }
    // for each user who has made a bid
on this product:
```

// get names and emails of all users who have made a bid on this product
$sql="SELECT * FROM BidEvents WHERE productID = '$productID'";
$result3 = $connection->query($sql);
if ($result3->num_rows>0) {
while ($row=$result3->fetch_assoc()) {
$bidderID = $row['buyerID'];
$sql = "SELECT * FROM Users WHERE userID = '$bidderID'";
$result4 = $connection->query($sql);
if($result4){
while ($row=$result4->fetch_assoc()) {
 	
$email_to_arr['bidderName'] = $row['username'];
 	
$email_to_arr['bidderEmail'] = $row['email'];
}
}
}
}		

// set ratings
function set_ratings($userID, $productID, $rating) {
 	
$sql = "INSERT INTO Ratings (userID, productID, rating_value) VALUES ($userID, $productID, $rating)";
$result = $connection->query($sql);
$connection->close();
return $array;
}
	When someone bids on an item, they can also choose to give the item a rating. This rating is then stored in a Ratings table.	6 (Recommend.ns) **probability_diff_interface.php**

function get_ratings($userID="") {
$array = [];
if (empty($userID)) {
 	
$sql = "SELECT DISTINCT productID FROM Ratings";
// find the items that are bought
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row = $result->fetch_assoc()) {
	Get ratings depending on the query - will be used for further processing.	6 (Recommend.ns) **probability_diff_interface.php**

array_push($array, $row["productID"]);
}
} else {
"Get ratings returns nothing";
}
 	
} else {
$sql = "SELECT DISTINCT productID FROM Ratings WHERE userID = '$userID'"; // find the items that are bought
$result = $connection->query($sql);
if ($result->num_rows>0) {
while ($row = $result->fetch_assoc()) {
array_push($array, $row["productID"]);
}
} else {
"Get ratings returns nothing";
}
}
 	
$connection->close();
return $array;
}		

//calculate popularity matrix in differential forms and stores it in
 	
function set_popularity_diff($some_arr, $instr) {
if (file_exists('../database.php')){
include '../database.php';
} else {
include './database.php';
}
$productID = $some_arr["productID"]; // item that was just rated
$userID = $some_arr["buyerID"]; // user that rated the item
 	
if ($instr === "insert") {
// get all of the users' rating pairs
$sql = "SELECT DISTINCT a1.productID, a2.rating_value - a1.rating_value as rating_diff
FROM Ratings as a1, Ratings as a2
WHERE a1.userID = '$userID' AND a2.userID = '$userID' AND a2.productID='$productID'";
$result = $connection->query($sql);
	Using ratings obtained from Ratings table, create a popularity matrix stored into popularity_diff table. This function calculates a relative rating between a product that was just rated, and every other product rated by the user ,and aggregated as a sum to be stored.
	
	It also records the ‘count’ of the number of times a product has been bidded on.
		6 (Recommend.ns) **probability_diff_interface.php**

```
        if ($result->num_rows>0) {
        // for every one of the users' rating
pairs, update popularity_diff table
        while
($row=$result->fetch_assoc()) {
        $other_productID =
$row["productID"];
        $rating_diff = $row["rating_diff"];

        // if the pair of products are already
in the popularity_diff table
        $sql = "SELECT productID1
FROM popularity_diff
                WHERE
productID1='$productID' AND
productID2='$other_productID'";

        $result_pair =
$connection->query($sql);

        if ($result_pair->num_rows>0) {
                // update the first row for
this pair
                $sql = "UPDATE
popularity_diff SET count=count+1,
sum=sum+$rating_diff
                WHERE
productID1='$productID' AND
productID2='$other_productID'";
                $result_update =
$connection->query($sql);

                // update the second row
for this pair
                // we only want to update if
the items are different
                if ($productID !=
$other_productID) {
                $sql = "UPDATE
popularity_diff SET count=count+1,
sum=sum-$rating_diff
                        WHERE
productID1='$other_productID' AND
productID2='$productID'";
                $result_update =
$connection->query($sql);

                }

        } else {
                $sql = "INSERT INTO
popularity_diff VALUES ('$productID',
'$other_productID', 1, '$rating_diff')";
                $result_insert =
$connection->query($sql);

                // insert second row for
```

this pair if they are not the same
if ($productID != $other_productID) {
$sql = "INSERT INTO popularity_diff VALUES ('$other_productID', '$productID', 1, -'$rating_diff')";
$result_insert = $connection->query($sql);
}
}
}		

function get_general_recommendations($current_productID, $n) {
$sql = "SELECT productID2, (sum/count) AS average
FROM popularity_diff
WHERE count > 1 AND productID1 = '$current_productID'
ORDER BY (sum/count) DESC
LIMIT $n";
$result = $connection->query($sql);
$list = [];
if ($result->num_rows>0) {
while ($row=$result->fetch_assoc()) {
$list[$row["productID2"]] = $row["average"];
}
}
return $list;
}
	From popularity_diff table, predict $n product that will be most highly rated relative to one product. Sum/count means, the better the item is compared to the rest (rating diff), the higher the sum, the more people bid on the same item, the higher the count, and sum/count acts like a normalisation. This is then used for every active product, and to choose the highest rated products predicted by these to act as non-personalised recommendations.	6 (Recommend.ns) **probability_diff_interface.php**

function get_personalised_recommendations($userID, $current_productID, $n) {//$current_productID) { //
$denom = 0.0; // denominator
$numer = 0.0; // numerator
$sql = "SELECT a.productID, a.rating_value
FROM Ratings a
WHERE a.userID=$userID AND a.productID != $current_productID";
$result = $connection->query($sql);
 	
// for all items that the user has
	From popularity_diff table, predict $n product that will be most highly rated for this user, and the selected product
	
	For each products that this user has rated, get an average rating between this and the selected product, based on what other users have rated.
		6 (Recommend.ns) **probability_diff_interface.php**

```
rated
        while
($row=$result->fetch_assoc()) {
        $fetched_productID =
$row["productID"];
        $ratings = $row["rating_value"];

        // get the number of times both
products have been rated by the user
        $sql = "SELECT pd.count,
pd.sum
                FROM popularity_diff pd
                WHERE
productID1=$current_productID and
productID2=$fetched_productID";
        $result_count =
$connection->query($sql);
        if ($result_count->num_rows>0)
{

        while ($row =
$result_count->fetch_assoc()) {
                $count = $row["count"];
                $sum = $row["sum"];

                // get average
                $average =
$sum/$count;

                $denom += $count;

                $numer += $count *
($average + $ratings);
                break;
        }
        }
        }
        $connection->close();

        if ($denom == 0) {
        return 0;
        } else {
        return ($numer/$denom);
        }

}
```