freeBay: An Online Auction Site

COMP0022 Coursework: Design Report*

December 2020

* The demonstration video can be found here.

Table of Contents

1 Background

freeBay is an online auction site designed in accordance with the requirements of COMP0022's postgraduate group coursework for the 2020/21 academic year.

This report outlines the design process for the database behind freeBay; a video demonstration of the site itself can be found here.

1.1 Project brief

Groups received a briefing document outlining the requirements of the coursework, including that it be built using WAMPServer (or equivalent) and not make use of any pre-made application frameworks (e.g. Django) – though frontend libraries could be used as desired.

freeBay implements each of the six target capabilities outlined in the briefing document (four 'core' and two 'extra'). These are reproduced below (with colour-coding for later use):

Figure 1: Six Target Capabilities

1. Users can register with the system and create accounts. Users have roles of seller or buyer with different privileges.
2. Sellers can create auctions for particular items, setting suitable conditions and features of the items including the item description, categorisation, starting price, reserve price and end date.
3. Buyers can search the system for particular kinds of item being auctioned and can browse and visually re-arrange listings of items within categories.
4. Buyers can bid for items and see the bids other users make as they are received. The system will manage the auction until the set end time and award the item to the highest bidder. The system should confirm to both the winner and seller of an auction its outcome.
5. Buyers can watch auctions on items and receive emailed updates on bids on those items including notifications when they are outbid.
6. Buyers can receive recommendations for items to bid on based on collaborative filtering (i.e., 'you might want to bid on the sorts of things other people, who have also bid on the sorts of things you have previously bid on, are currently bidding on).

1.2 Assumptions

Since there is no 'client' to refine the system and its design with, several important assumptions were made during the design process regarding both the client's desires in the present moment and how these might change in the future. These are highlighted throughout the document as they are made.

For example, the assumption was made that the client does not wish to allow users to be both a buyer and a seller, which is the most direct interpretation of capability 1. This is a reasonable assumption: the client might wish to have *only* registered businesses as sellers, thereby providing 'Amazon-like' reliability in an 'eBay-like' auction system.

The guiding principles behind freeBay's database design were that it should:

- Incorporate all of the most important features for the site to function well *today*
- Have the flexibility to include *future* features when allowing this flexibility does not require a radically different approach today

1.3 Report structure and approach

This report follows a simplified version of the database design methodology outlined in Chapters 16-18 of Connolly and Begg (2015)[1].

It is simplified in the sense that it combines certain logically related sections (e.g. relating to defining attributes) and re-orders others (e.g. normalization to 1NF *during* the translation of the ER diagram) so that it can be described more concisely.

The first two stages relate to 'conceptual' and 'logical' design. The logical design section, in particular, contains discussions of various subtleties, including issues around:

- *Null values* and 1NF
- *Referential integrity*

This is followed by a shorter section on 'physical design', since the database system itself was known since the beginning of the project (MySQL[2]). This section focuses on:

- *Integrity constraint* enforcement in the database
- Issues around *derived data* and denormalization

Finally, the database queries supporting each capability are listed and their purposes explained. For more complex queries (e.g. in capability 6), the rationale is also given.

[1] Connolly, T. and Begg, C., 2015. *Database Systems*. 6th ed. Boston: Pearson Education.
[2] The specific version used was MySQL 8.0.22

2 Conceptual Design

This section details the *conceptual* design of the database: the design of a data model based on the six capabilities listed in Figure 1 without consideration of how this model would (best) be implemented as a relational schema or in specific versions of MySQL.

2.1 Identifying entities and relationships

After a close examination of Figure 1, several entities (blue) and relationships (orange) can be identified. The entities are *Users* (composed of *Buyers* and *Sellers*), *Auctions* and *Bids*, with the following relationships between them[3]:

- Buyers and Sellers are subclasses of the Users superclass {Mandatory, Or}
- (1..1) Sellers create Auctions (0..*)
- Buyers place Bids on Auctions (multiplicities given in ER diagram)
- (0..*) Buyers watch Auctions (0..*)

Multiplicities for the relationship are given next to the corresponding entities, but for these multiplicities there is little flexibility (e.g. buyers are not compelled to bid on auctions).

The most significant assumption here is that the parent/child relationship between Users and Buyers/Sellers is {Mandatory, Or}, the motivation for which was given in section 1.2. This is an important assumption since it has implications for the database's relational schema, and is therefore returned to once again in section 3.

Whilst 'categorisation' can be thought of as an attribute (green) of auctions, it would be better to think of it as a separate entity. This is because the site should provide sellers with a *common* list (populated from a database) of categories to choose from: this both increases usability for sellers and enables buyers to efficiently filter auctions by category (capability 3). We therefore have an additional entity (*Categories*) and relationship:

- (0..*) Auctions have Categories (1..1)

The decision was made here to only allow auctions to have one category, rather than multiple categories, since an auction requiring 'multiple' categories would suggest that the categorisation is not completely MECE.

2.2 Identifying attributes

It is also possible to identify several entity attributes (green) from Figure 1, but only for the 'auctions' entity: *description*, *starting price*, *reserve price* and *end date/time*.

---

[3] User registration (capability 1), auction search (capability 3) and recommendations (capability 6) represent interactions with the database, but not relationships between entities themselves.

In reality, auctions and the other entities could each have a wide variety of attributes. The criteria used for determining which attributes to include per entity were:

- *Necessity:* Necessary based on the capabilities required (e.g. reservePrice) or necessary in general for an auction site (e.g. address)
- *Completeness:* Supporting a useful additional feature that is straightforward to implement in a complete way (e.g. minIncrement)

The attributes chosen per entity, and their domains, are listed in Figure 2 below. Attributes allowing (unlimited) multiple values are marked with '⁂', and none are derived:

Figure 2: Included attributes per entity

Entity	Attribute	Domain	Optional?
Users	username {CK}	20 variable characters	N
	email {CK}	254 variable characters	N
	pass(word)	60 characters (hashed)	N
	name		
	firstName	35 variable characters	N
	familyName	35 variable characters	N
	address ⁂		Y
	line1	35 variable characters	
	city	35 variable characters	
	postcode	35 variable characters	
	country	35 variable characters	
	telNo ⁂	15 variable numbers	Y
Bids	bidDate	Timestamp (incl. seconds)	N
	bidAmount	GBP amount (incl. pennies)	N
Auctions	itemDetails		
	title	80 variable characters	N
	descript(ion)	4000 variable characters	N
	dates		
	createDate	Timestamp (incl. seconds)	N
	startDate	Timestamp (incl. seconds)	N
	endDate	Timestamp (incl. seconds)	N
	financials		
	startPrice	GBP amount (incl. pennies)	N
	reservePrice	GBP amount (incl. pennies)	Y
	minIncrement	GBP amount (incl. pennies)	Y
Categories	categoryName {CK}	35 variable characters	N

The *username* attribute was included for users since auction sites typically list auctions' highest bidders' and sellers' usernames on listing pages. Whilst email addresses can be used to uniquely identify users in the system, they should not be shared publicly.

Addresses were also included for users as most users will need an address at some point (for receiving goods or returns), and often more than one (e.g. a billing and delivery address). Auction sites also sometimes give sellers the option to share an (official) *telephone number* with the winning bidder of their auction(s). These could also be useful for future account security features (e.g. two-factor authentication). Both addresses and telephone numbers, however, should be optional at the registration stage.[4]

Auctions' *reserve prices* should be optional as some sellers do not wish to include these (they might just want the item to sell if it receives any bids), and some bidders prefer to bid on auctions without a reserve price (since the reserve price is not visible to them).

An optional *minimum increment* attribute was also included for auctions: this is a commonly used tool in auction sites to help the auction's current price increase earlier – rather than allowing tactical, last-minute out-bidding by very small amounts.

Attribute domains were chosen in a variety of ways. For some, the technical upper limit was used (e.g. international telephone numbers can be up to 15 digits, email addresses can be a maximum of 254 characters), and for others guidelines were taken from third parties:

- *title, description:* Same domains as eBay
- *categoryName:* eBay's top-level categories were used in the database, each of which are less than 35 characters long.
- *name, address:* Follow the UK government guidelines for its own services[5]

2.3 Determining primary keys

The *Users* entity has two candidate keys already: *username* and *email*. These are unique for both security reasons (e.g. account recovery) and for identifying users on listing pages, as discussed previously. However, a separate *userID* primary key should be added so that users can change their username or email in the future without issue.

Similarly, *categoryID* should be added to *Categories* as its primary key so that categories can be changed in the future in a consistency-preserving way without issue (e.g. changing a current category to a superclass of it).

---

[4] Whilst more than one address and telephone are allowed in theory, the freeBay site currently only supports users adding up to one of each during registration.

[5] webarchive.nationalarchives.gov.uk. n.d. [online] Available at: <https://webarchive.nationalarchives.gov.uk/+/http://www.cabinetoffice.gov.uk/media/254290/GDS%20Catalogue%20Vol%202.pdf> [Accessed 25 November 2020].

Auctions can be given an *auctionID* primary key (despite being a weak entity) at this stage since it doesn't currently have a candidate key. This is because auction titles are not enforced to be unique: eBay does the same. It will eventually need a primary key, and it's simpler to identify an auction by an ID than a composite of its attributes and *userID* (of the seller).

Similarly, a *bidID* primary key can be added to *Bids* at this stage for simplicity's sake (despite Bids also being a weak entity) because it will eventually need a primary key, and it's simpler to identify a bid by an ID than a composite of its attributes, *userID* (of the buyer) and *auctionID*.

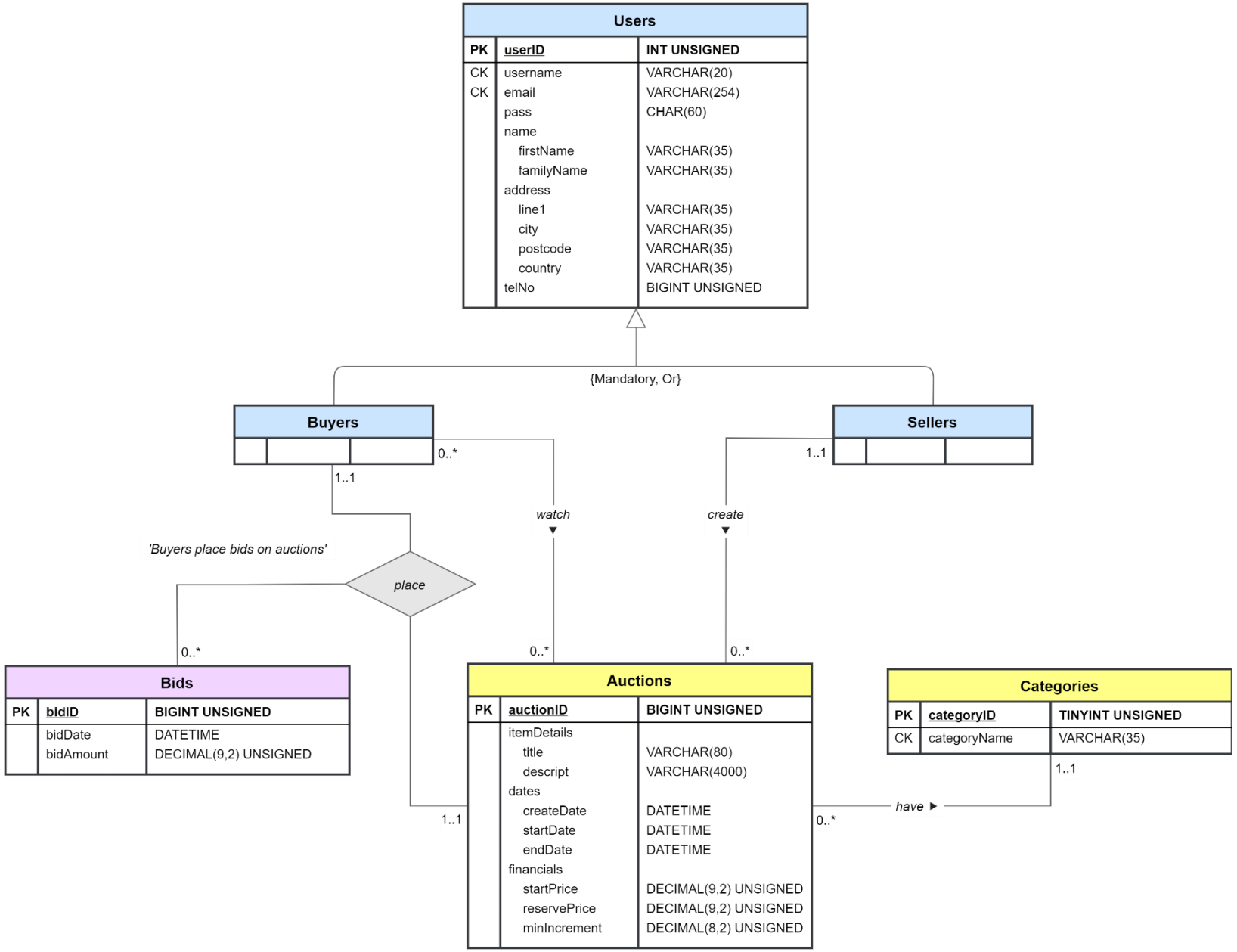
2.4 Entity-relationship (ER) diagram

The entity relationship (ER) diagram for this set of entities, relationships and attributes is given in Figure 3. The ER diagram contains the MySQL-equivalent of the domains given in Figure 2, with limits imposed on how high a bid/price can reasonably be[6].

The diagram makes it clear that there are no redundant entities or relationships, and no fan or chasm traps have inadvertently been created.

[6] minIncrement is set as one significant figure smaller than other prices since this reflects the differences between prices, not their final (absolute) values.

Figure 3: Entity-relationship (ER) diagram



3 Logical Design

This section details the *logical* design of the database: the process of mapping the previous section's conceptual design to a logical design aligned, in this case, with the relational model (since MySQL is a relational database management system).

3.1 Deriving relations from the conceptual design

Figure 4 below describes a sequence of derivations translating Figure 3's ER diagram to a complete (relational) database schema (section 3.2) in a concise way:

Figure 4: Step-by-step derivation of relations

Step	**Analysis**	**Implications**
1.Superclass/ subclass relationships	• Could have one relation per subclass (Buyers, Sellers) or a combined relation with an extra 'type' attribute • If there are no plans to allow multiple roles in the future, better to create one relation each – though this makes the schema and some queries more complex (e.g. checking email uniqueness)	• Users are represented by two relations: *Buyers* and *Sellers* • These have *buyerID* and *sellerID* primary keys respectively • Composite attributes flattened
2. Preventing null values for optional attributes	• There is debate over whether 1NF should allow nulls. A 'mathematical' relation must have one entry per 'cell', and nulls make queries more complex since they can have unexpected behaviour (e.g. in joins). Decision taken to *prevent nulls* on both theoretical and practical grounds. • For addresses/telNos, create separate relations. Treat these as 1:* rather than *:* - whilst users can share these by coincidence, a change in one user's value (e.g. address) does not necessarily imply a change in all users sharing that value (e.g. a student moving to university). • Create a separate *Countries* relation since country names are known, and it can be used to populate front-end widgets • For *reservePrice* and *minIncrement,* set these equal to startPrice and £0.01 by default respectively (in PHP)	• Five new relations – these cannot be combined since a FK can only point to one table: ○ *BuyerAddresses* ○ *BuyerTels* ○ *SellerAddresses* ○ *SellerTels* ○ *Countries* • 'Address' relations given *addressID* PKs for simplicity • Countries relation given *countryID* PK for simplicity and in case countries change names (happens occasionally) • 'Tel' relations do not require a dedicated PK

3. Other strong entities	• Categories is the remaining 'strong' entity not yet considered, and should be given its own relation	• New *Categories* relation. PK already identified.
4. Weak entities	• Bids and Auctions are both 'weak' entities, and should be given their own relations	• New *Bids* and *Auctions* relations. PKs already identified. • Composite atts. flattened

5. 1: relationships*	• Numerous 1:* relationships identified
	• The '1' side of the relationship is designated as the parent
	• The '*' side gets the PK of the parent as a FK
		• *buyerID* FK added to:
		  ○ BuyerAddresses
		  ○ BuyerTels
		• *sellerID* FK added to:
		  ○ SellerAddresses
		  ○ SellerTels
		  ○ Auctions
		• *countryID* FK added to:
		  ○ BuyerAddresses
		  ○ SellerAddresses
		• *categoryID* FK added to:
		  ○ Auctions

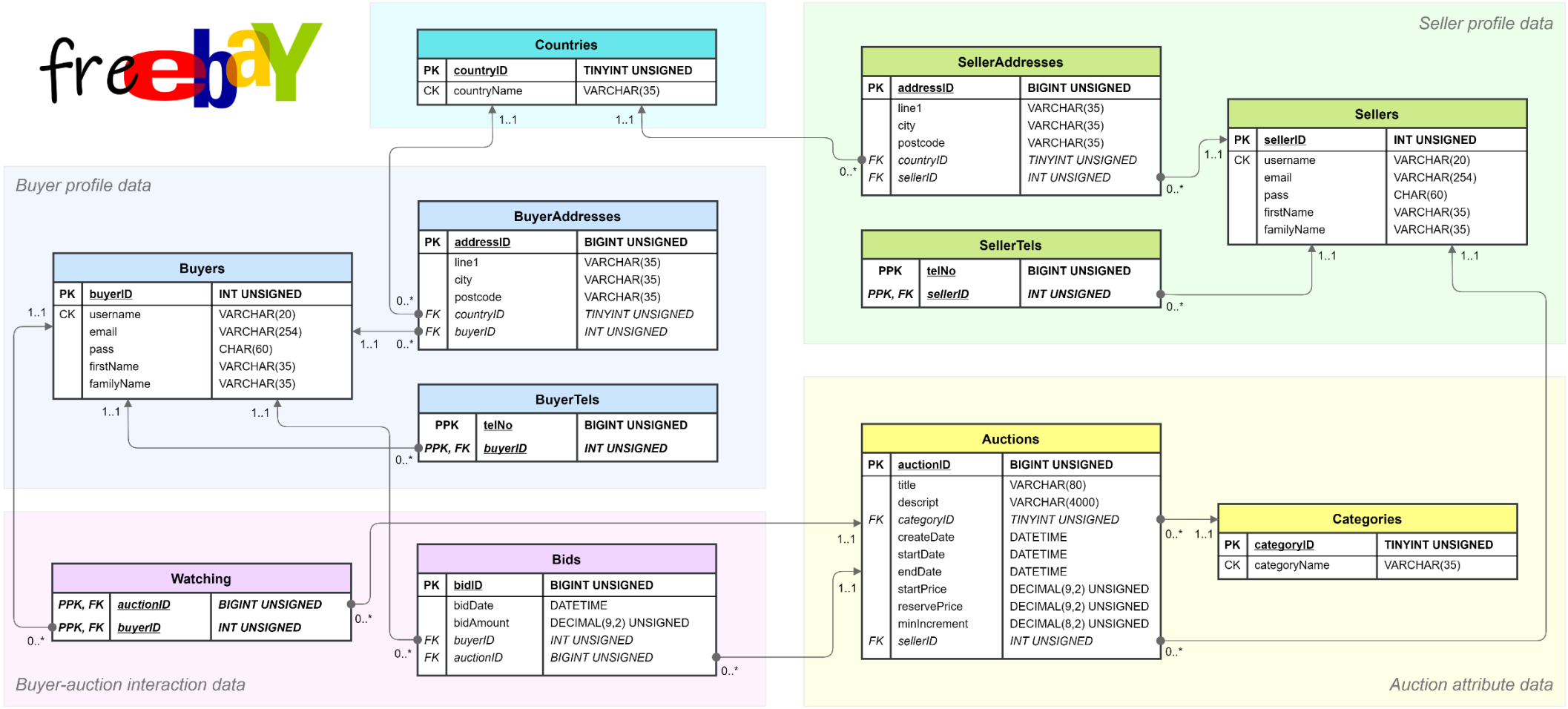
*6. *:* relationships*	• Relevant to the 'Buyers *watch* auctions' relationship • Create a dedicated relation to represent the relationship, with PKs of both entities added as FKs to the new relation	• New *Watching* relation • No need for a dedicated PK • *buyerID* and *auctionID* added as FKs
7. Complex relationship types	• Relevant to the 'Buyers *place* Bids *on* Auctions' relationship • Since Bids is already its own relation (with a PK), and it has the many ('*') cardinality, the PKs of Buyers and Auctions should be added to it as FKs	• *buyerID* and *auctionID* added as FKs to the Bids relation

3.2 Database schema

The net result of the derivations in section 3.1 are set of eleven relations satisfying 3NF. These are first shown graphically in Figure 5 before being listed, where underlined attributes are (partial) primary keys, and italicised attributes are foreign keys.

Note that whilst most relations have a dedicated primary key, *BuyerTels*, *SellerTels* and *Watching* have partial primary keys.

Figure 5: Database schema



1. **Buyers** (buyerID, username, email, pass, firstName, familyName)
2. **BuyerAddresses** (addressID, line1, city, postcode, *countryID, buyerID*)
3. **BuyerTels** (telNo, *buyerID*)
4. **Sellers** (sellerID, username, email, pass, firstName, familyName)
5. **SellerAddresses** (addressID, line1, city, postcode, *countryID, sellerID*)
6. **SellerTels** (telNo, *sellerID*)
7. **Countries** (countryID, countryName)
8. **Bids** (bidID, bidDate, bidAmount, *buyerID, auctionID*)
9. **Watching** (*auctionID*, *buyerID*)
10. **Auctions** (auctionID, title, descript, createDate, startDate, endDate, startPrice, reservePrice, minIncrement, *sellerID, categoryID*)
11. **Categories** (categoryID, categoryName)

3.3 Validating relations using normalization

To verify that the database schema is in third normal form (3NF), it is sufficient to show that, for each relation, *no non-candidate key attribute is transitively dependent on any candidate key.*

The functional dependencies for each relation are listed in Figure 6 below, along with their associated type. Transitive dependencies that intuitively appear possible but in reality do *not* hold are also included in grey (and discussed afterwards). If there is a functional dependency of type 'transitive', the database schema violates 3NF:

Figure 6: Identifying functional dependencies

Relation(s)	Functional Dependency	Type
Buyers, *Sellers*	buyerID/sellerID Ⓡ pass, firstName, familyName username Ⓡ pass, firstName, familyName email Ⓡ pass, firstName, familyName	Primary key Candidate key Candidate key
BuyerAddresses, *SellerAddresses*	addressID Ⓡ line1, city, postcode, countryID, sellerID line1, city, countryID Ⓡ postcode line1, postcode, countryID Ⓡ city	Primary key False False
Auctions	auctionID Ⓡ title, descript, createDate, startDate, endDate, startPrice, reservePrice, minIncrement, sellerID, categoryID createDate Ⓡ startDate startDate Ⓡ endDate startPrice Ⓡ reservePrice	Primary key False False False

Bids	bidID Ⓡ bidDate, bidAmount, buyerID, auctionID	Primary key
BuyerTels, SellerTels, Watching	No non-candidate-key attributes (except the PPKs)	-
Countries, Categories	No non-candidate-key attributes	-

Since there are no 'Transitive' entries in the 'Type' column, we can conclude that the database schema is in 3NF. Several 'candidate' transitive dependencies were identified, however, and the reasons for dismissing them are as follows:

- *line1, city, countryID* Ⓡ *postcode:* In the UK, a city can have multiple streets with the same name (and overlapping numbering, i.e. line1), but in different parts of the city – and therefore with different postcodes.
- *line1, postcode, countryID* Ⓡ *city:* In the UK, some cities share the same postcode system (e.g. Basingstoke and Reading both have 'RG' prefixes).
- *createDate* Ⓡ *startDate:* Sellers are given the option to set their auction to start at some point in the future.
- *startDate* Ⓡ *endDate:* Sellers can choose auctions of different lengths.
- *startPrice* Ⓡ *reservePrice:* Sellers are given the option to include a reserve price or not, so this doesn't hold true in general.

3.4 Defining integrity constraints

There are several types of integrity constraint that could be imposed in order to ensure the database remains complete, accurate and internally consistent. Some of these have already been discussed:

- *Required data:* Null values are not allowed in the database.
- *Attribute domain constraints:* Domains were given in Figure 5.
- *Multiplicity:* Multiplicity constraints were given in Figure 5.
- *Entity integrity:* Primary keys cannot contain nulls (this will be implemented automatically by MySQL).

Two further types of constraint are *referential integrity* constraints (conditions under which a foreign key may be inserted, updated or deleted) and *general* constraints (reflecting considerations about the realities of an online auction system). These are discussed in turn.

3.4.1 Referential integrity constraints

In total, the database schema contains twelve foreign keys, and there are four main events to consider for each:

1. *Insertion of tuple into child relation:* The FK attribute of the new child tuple should match a PK in the parent relation.
    - MySQL (InnoDB) does this automatically
2. *Update of FK in child tuple:* The new FK attribute in the child tuple should match a PK in the parent relation
    - MySQL (InnoDB) does this automatically
3. *Update of PK in parent tuple:* If a PK attribute is updated in a parent tuple, these should be cascaded to all corresponding tuples in the child relation
    - MySQL (InnoDB) allows this to be specified by '`ON UPDATE CASCADE`'
4. *Deletion of tuple from parent relation:* There are different strategies here, which are appropriate in different circumstances. They are important to consider, however, due to personal data laws (deletion should be allowed of user data). The option chosen for each FK, and a rationale, are given in Figure 7.
    - MySQL (InnoDB) *does not natively support* all possible strategies here.

Figure 7: Referential integrity – 'On delete...'

Foreign Key(s)	**Constraint**	**Rationale**
BuyerAddresses *buyerID* SellerAddresses *sellerID*	`ON DELETE CASCADE`	If a user is deleted, no need for their address data
BuyerTels *buyerID* SellerTels *sellerID*	`ON DELETE CASCADE`	If a user is deleted, no need for their telephone data
BuyerAddresses *countryID* SellerAddresses *countryID*	`ON DELETE NO ACTION`	Most likely a mistake
Bids *buyerID*	`ON DELETE SET DEFAULT`	*See below*
Bids *auctionID*	`ON DELETE CASCADE`	If an auction is deleted, no need for historic bid data
Watching *buyerID* Watching *auctionID*	`ON DELETE CASCADE`	If an auction is deleted, no need for historic watch data
Auctions *sellerID*	`ON DELETE SET DEFAULT`	*See below*
Auctions *categoryID*	`ON DELETE SET DEFAULT`	*See below*

The situation for the Bids *buyerID* FK, and the Auctions *sellerID* and *categoryID* FKs is slightly more complex. In each case, we should allow the deletion of buyers, sellers and categories, but we should not also delete the bid or auction data since this is still useful:

- Completed auctions are frequently studied by sellers to decide how to price and pitch their own, similar, auction, and by buyers to decide how much to bid on similar auctions.
- Bid information must therefore be preserved too, since this determines whether an auction sold and, if it did, its sale price.
- Categories might also change with time, especially if the auction site decides to focus on specific types of item

Ideally, each would be set to a 'default' generic value on deletion of their parent tuple:

- Bids *buyerID* to 'N/A'
- Auctions *sellerID* to 'N/A'
- Auctions *categoryID* to 'Other'

A further complication here is that MySQL does not support '`ON DELETE SET DEFAULT`'. Section 4.1 describes how this was accomplished in practice.

3.4.2 General constraints

There are also six important *general* constraints to impose when data is inserted into the database, for both security and consistency reasons:

1. startDate should be *weakly later* than createDate. Implemented in MySQL during Auction table creation with:
    - `CONSTRAINT CHK_startDate CHECK (startDate >= createDate)`
2. endDate should be *strictly later* than startDate. Implemented in MySQL during Auction table creation with:
    - `CONSTRAINT CHK_endDate CHECK (endDate > startDate)`
3. reservePrice should be *weakly greater* than startPrice. Implemented in MySQL during Auction table creation with:
    - `CONSTRAINT CHK_reservePrice CHECK (reservePrice >= startPrice)`
4. username should be *unique* across *both* buyers and sellers
5. email should be *unique* across *both* buyers and sellers
6. bidAmount should be *greater than* (current highest bid + minIncrement) or startPrice if there have been no bids

The final three constraints are more complex to implement in MySQL. Section 4.2 describes how this was accomplished in practice.

4 Physical Design

This section details the *physical* design of the database: how elements of the logical design were implemented in practice in MySQL.

Previous sections have already given details on, for example, attribute data types in MySQL and the implementation of various constraints. The sub-sections below provide more detail on elements for which the logical design is not straightforward to implement in MySQL.

4.1 Referential integrity constraints

As described in section 3.4.1, it is desirable to have '`ON DELETE SET DEFAULT`' logic for the Bids *buyerID* FK, and the Auctions *sellerID* and *categoryID* FKs. MySQL (InnoDB) does not, however, currently support this.

It is nevertheless possible to achieve the same effect through a combination of:

1. A new *Defaults* relation with '`ON DELETE NO ACTION`'
2. One trigger per FK
3. '`ON DELETE CASCADE`' for the three FKs

Firstly, the Buyers, Sellers and Categories relations are each given a 'dummy' record containing the following data (they are inserted first into the database):

- *Buyers:* buyerID = 1, username = 'N/A' (other attribute values arbitrary)
- *Sellers:* sellerID = 1, username = 'N/A' (other attribute values arbitrary)
- *Categories:* categoryID = 1, categoryName = 'Other'

Next, a new *Defaults* relation is created with one record and three attributes[7]:

- buyerID = 1, sellerID = 1, categoryID = 1

Each of the three attributes are given FK constraints on the corresponding relations with '`ON DELETE NO ACTION`' specified to prevent deletion of these 'default' values:

A trigger is then created that updates the child tuples' FK values to 1 before the deletion of the parent tuple. This ensures that referential integrity is preserved.

Finally, '`ON DELETE CASCADE`' is applied to the child tuple: since the parent tuple no longer has any child tuples, this is safe to do. Whilst '`ON DELETE NO ACTION`' would have the same effect here, we don't want to prevent deletion of parent tuples in general, for the reasons discussed previously.

[7] The 'Admin' user is not given permissions on this table to stop them interacting with it (only root can).

Full details of the implementation can be found in the `private/database_generator.sql` script. For example, the trigger used for the Bids *buyerID* FK is:

```
DELIMITER $$
CREATE TRIGGER Auctions_fk1_trigger
    BEFORE DELETE ON Sellers FOR EACH ROW
    BEGIN
    UPDATE Auctions
        SET sellerID = 1 WHERE sellerID = OLD.sellerID;
    END $$
DELIMITER ;
```

4.2 General constraints

As described in section 3.4.2, there are three important *general* constraints to impose when data is inserted into the database that cannot be implemented directly in MySQL.

Whilst these could be implemented in PHP alone, it is good practice to enforce the most important constraints in the database too as, in general, security and data validation should be implemented in 'layers', and new applications might, in the future, use the database too (e.g. a mobile app). Moreover, SQL is a declarative rather than procedural language – potentially making the constraints easier to define without error.

In MySQL, the same effect can be accomplished through a combination of stored functions and triggers:

1. username should be *unique* across *both* buyers and sellers
    - Function: `username_check` (checks username is unique)
    - Triggers (call the function):
        - `CHK_buyer_username_unique` (before insert)
        - `CHK_seller_username_unique` (before insert)
2. email should be *unique* across *both* buyers and sellers
    - Function: `email_check` (checks email is unique)
    - Triggers (call the function):
        - `CHK_buyer_email_unique` (before insert)
        - `CHK_seller_email_unique` (before insert)
3. bidAmount should be *greater than* (current highest bid + minIncrement) or startPrice if there have been no bids
    - Function: `bid_check` (checks bid is sufficiently high)
    - Trigger (calls the function):
        - `CHK_bid_valid` (before insert)

Full details of the implementation can be found in the `private/database_generator.sql` script. For example, the relevant code for checking bidAmount is:

```
DELIMITER $$
CREATE FUNCTION bid_check (
  amount DECIMAL(9,2) UNSIGNED,
  auction BIGINT UNSIGNED
  )
  RETURNS VARCHAR(7)
  BEGIN
    DECLARE storeVal1 DECIMAL(9,2) UNSIGNED;
    DECLARE storeVal2 DECIMAL(9,2) UNSIGNED;
      SELECT COUNT(bidID) INTO storeVal1 FROM Bids WHERE auctionID = auction;
      IF storeVal1 = 0 THEN
        SELECT startPrice INTO storeVal2 FROM Auctions WHERE auctionID = auction;
        IF amount >= storeVal2 THEN
          RETURN 'valid';
        ELSE
          RETURN 'invalid';
        END IF;
      ELSE
        SELECT MAX(bidAmount) INTO storeVal1 FROM Bids WHERE auctionID = auction;
        SELECT minIncrement INTO storeVal2 FROM Auctions WHERE auctionID = auction;
        IF amount >= (storeVal1 + storeVal2) THEN
          RETURN 'valid';
        ELSE
          RETURN 'invalid';
        END IF;
      END IF;
  END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER CHK_bid_valid
  BEFORE INSERT ON Bids FOR EACH ROW
  BEGIN
    DECLARE storeVal VARCHAR(7);
    SET storeVal = bid_check(NEW.bidAmount, NEW.auctionID);
    IF storeVal = 'invalid' THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = "Please enter a valid bid amount.";
    END IF;
  END $$
DELIMITER ;
```

4.3 Derived data

As it stands, the database does not contain any derived attributes or data. There are, however, several derived attributes related to an auction that could naturally and usefully be attached to it, including:

- Time remaining
- Current highest bid
- Current highest bidder
- Number of bids
- Number of watchers
- Reserve price met (Y/N)

Moreover, all of these except 'time remaining' might offer a performance and efficiency advantage if they were stored, since they would usually be updated less frequently than they are accessed by the website front-end – the calculation could be done in advance[8].

However, derived data goes against the spirit of normalization, which requires an attribute's value to only be knowable in 'one way'. While some of these attributes might not be functionally determined by any attributes in the Auctions relation itself (e.g. number of bids) and, therefore, by some definitions, fail to violate 3NF on a technicality, the basic argument in favour of normalization stands.

Ultimately, any decision to 'denormalize' the database on the grounds of performance would need to be thought out carefully, and the advantages clearly quantified.

[8] Response times are critical in eCommerce applications. For example, research by Google in 2016 found that 53% of mobile website visitors leave a webpage if it doesn't load within three seconds.

5 Database Queries

This section lists the queries used in the PHP code underpinning freeBay. For more complex queries, the rationale behind their design is also given.

5.1 Capability 1

register.php

- Populates the registration form's country dropdown in alphabetical order

```
SELECT countryName FROM Countries ORDER BY countryName ASC
```

php/check_email.php

- Run by an AJAX call in register.php to check the email is unique in real time

```
SELECT email_check('$email')
```

php/check_username.php

- Run by an AJAX call in register.php to check the username is unique in real time

```
SELECT username_check('$username')
```

process_registration.php

For buyers:

- Inserting required fields into the Buyers table

```
INSERT INTO Buyers (username, email, pass, firstName, familyName)
VALUES ('$username', '$email', '$passwordHash', '$firstName', '$familyName')
```

- Inserting (optional) address data into the BuyerAddresses table

```
INSERT INTO BuyerAddresses (line1, city, postcode, countryID, buyerID)
VALUES ('$line1',
       '$city',
       '$postcode',
       (SELECT countryID FROM Countries WHERE countryName = '".$country."'),
       (SELECT buyerID FROM Buyers WHERE username = '".$username."'))
```

- Deleting the required fields from Buyers table if the address insertion query failed

```
DELETE FROM Buyers WHERE username = '".$username."'
```

- Inserting (optional) telephone number data into the BuyerAddresses table

```
INSERT INTO BuyerTels (telNo, buyerID)
VALUES ('$telNo',
        (SELECT buyerID FROM Buyers WHERE username = '".$username."'))
```

- Deleting the required fields from Buyers table if the telNo insertion query failed

```
DELETE FROM Buyers WHERE username = '".$username."'
```

- Deleting address data from the BuyerAddresses table if the telNo insertion query failed, and the user had inputted an address too.

```
DELETE FROM BuyerAddresses
WHERE buyerID = (SELECT buyerID FROM Buyers WHERE username = '".$username."')
```

For sellers (equivalent queries to the buyer queries in this section):

- Inserting required fields into the Sellers table

```
INSERT INTO Sellers (username, email, pass, firstName, familyName)
VALUES ('$username', '$email', '$passwordHash', '$firstName', '$familyName')
```

- Inserting (optional) address data into the SellerAddresses table

```
INSERT INTO SellerAddresses (line1, city, postcode, countryID, sellerID)
VALUES ('$line1',
       '$city',
       '$postcode',
       (SELECT countryID FROM Countries WHERE countryName = '".$country."'),
       (SELECT sellerID FROM Sellers WHERE username = '".$username."'))
```

- Deleting the required fields from Sellers table if the address insertion query failed

```
DELETE FROM Sellers WHERE username = '".$username."'
```

- Inserting (optional) telephone number data into the SellerAddresses table

```
INSERT INTO SellerTels (telNo, sellerID)
VALUES ('$telNo',
        (SELECT sellerID FROM Sellers WHERE username = '".$username."'))
```

- Deleting the required fields from Sellers table if the telNo insertion query failed

```
DELETE FROM Sellers WHERE username = '".$username."'
```

- Deleting address data from the SellerAddresses table if the telNo insertion query failed, and the user had inputted an address too.

```
DELETE FROM SellerAddresses
WHERE sellerID = (SELECT sellerID FROM Sellers WHERE username = '".$username."')
```

login_result.php

- Retrieving the user's password hash from the database for authentication

```
SELECT pass FROM Buyers WHERE username = '".$username."'
UNION ALL
SELECT pass FROM Sellers WHERE username = '".$username."'
```

- Retrieving the user's ID for storing as a session variable

```
SELECT buyerID AS userID
FROM (SELECT buyerID, username FROM Buyers
      UNION ALL
      SELECT sellerID, username FROM Sellers) AS Combined
WHERE username = '".$username."'
```

- Checking if the user is a buyer (for storing as a session variable)

```
SELECT 1 FROM Buyers WHERE username = '".$username."'
```

- Checking if the user is a seller (for storing as a session variable)

```
SELECT 1 FROM Sellers WHERE username = '".$username."'
```

5.2 Capability 2

create_auction.php

- Populates the auction creation form's category dropdown in alphabetical order

```
SELECT categoryName FROM Categories ORDER BY categoryName ASC
```

create_auction_result.php

- Inserts auction data into the database

```
INSERT INTO Auctions (title, descript, createDate, startDate, endDate,
                      startPrice, reservePrice, minIncrement, sellerID,
                      categoryID)
VALUES ('$title', '$descript', '$createDate', '$startDate', '$endDate',
        '$startingPrice', '$reservePrice', '$minIncrement', '$sellerID',
        (SELECT categoryID FROM Categories WHERE categoryName = '$category'))
```

mylistings.php

This file contains all of the queries in `index.php` (the main auction listing page), though with slightly different filters. For example, it allows filtering for the seller's auctions that *will sell* (if they have a reserve price, this must have been met, otherwise, they must have received at least one valid bid).

It also contains one extra query:

- Determines the current state of the auction, including the username of the current highest bidder

```
SELECT a.auctionID, b.bidAmount, a.reservePrice, b.buyerID, bu.username
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
JOIN Buyers bu ON b.buyerID = bu.buyerID
WHERE a.auctionID = '$productID'
AND b.bidAmount = (SELECT MAX(bidAmount)
                   FROM Bids
                   WHERE auctionID = '$productID')
```

5.3 Capability 3

index.php

- Retrieving the categories from the database to filter listings by

```
SELECT categoryName, categoryID FROM Categories
```

- The first part of the main query: collects information to display in the auction cards

```
$sql =
SELECT mainTable.*, categoryName
FROM ($sqlMainMaxBidWatchSeller) mainTable
INNER JOIN Categories ON mainTable.categoryID = Categories.categoryID
```

To this, simple `WHERE` *conditions are concatenated before the query is called based on the filters chosen and search terms entered by the user. For example, if the user enters a search term and then filters for auctions in the category 'Fashion' (ID = 4), which are completed (and sold successfully), ordered by how newly they were listed (descending), this code will be appended:*

```
WHERE Categories.categoryID IN ('4')
AND title LIKE '%$searchToken%'
AND '$currentTime' > endDate
AND (maxBid >= reservePrice AND noOfBidders > 0)
ORDER BY createDate DESC
```

Full details can be found in lines 242-304 of the file.

Five simpler queries are stored as strings to assist in building the main query. Only one query, the main query, is called:

- Retrieves all relevant auction information, paired with the seller's username

```
$sqlMainMaxBidWatchSeller =
SELECT mainTable.*, username AS sellerUsername
FROM ($sqlMainMaxBidWatch) mainTable
INNER JOIN Sellers ON mainTable.sellerID = Sellers.sellerID
```

- Retrieves relevant auction information (details + bids) incl. the number of watchers

```
$sqlMainMaxBidWatch =
SELECT mainTable.*, COALESCE(noOfWatching, 0) AS noOfWatching
FROM ($sqlMainMaxBid) mainTable
LEFT JOIN ($sqlWatchTable) watchTable ON mainTable.auctionID = watchTable.auctionID
```

- Retrieves the number of watchers for each auction

```
$sqlWatchTable =
SELECT COUNT(auctionID) AS noOfWatching, auctionID
FROM Watching
GROUP BY auctionID
```

- Retrieves relevant auction information (details + bid-related), including the 'effective' maximum bid and the 'effective' number of bidders

```
$sqlMainMaxBid =
SELECT mainTable.*,
       COALESCE(maxBid, mainTable.startPrice) AS maxBid,
       COALESCE(noOfBidders, 0) AS noOfBidders,
FROM Auctions mainTable
LEFT JOIN ($sqlMaxBid) maxBidTable ON mainTable.auctionID = maxBidTable.auctionID
```

- Retrieves the maximum bid and number of bidders for each auction

```
$sqlMaxBid =
SELECT MAX(bidAmount) AS maxBid,
       COUNT(DISTINCT buyerID) AS noOfBidders,
       auctionID
FROM Bids
GROUP BY auctionID
```

5.4 Capability 4

listing.php

- Retrieves all the relevant summary information about the auction

```
SELECT a.title, a.descript, a.startDate, a.endDate, a.startPrice,
       a.reservePrice, a.minIncrement, c.categoryName, s.username,
       s.sellerID, IFNULL(p.maxBid, 0) AS maxBid,
       IFNULL(p.numBids, 0) AS numBids, IFNULL(w.watchers, 0) AS watchers
FROM Auctions a
LEFT JOIN Categories c ON a.categoryID = c.categoryID
LEFT JOIN Sellers s ON a.sellerID = s.sellerID
LEFT JOIN ($sql_1) p ON p.auctionID = a.auctionID
LEFT JOIN ($sql_2) w ON w.auctionID = a.auctionID
WHERE a.auctionID = '$auction_id'
```

where the two below queries are stored as strings (not called separately):

- Retrieving the auction's current price and number of bids

```
$sql_1 =
SELECT auctionID, MAX(bidAmount) AS maxBid, COUNT(bidID) AS numBids
FROM Bids
WHERE auctionID = '$auction_id'
GROUP BY auctionID
```

- Retrieving the auction's current number of watchers

```
$sql_2 =
SELECT auctionID, COUNT(buyerID) AS watchers
FROM Watching
WHERE auctionID = '$auction_id'
GROUP BY auctionID
```

- Retrieving the auction's bid history (for putting in a table)

```
SELECT bidDate, username, bidAmount
FROM Bids, Buyers
WHERE bids.buyerID = buyers.buyerID AND auctionID = '$auction_id'
ORDER BY bidDate
```

- Determining if the user is already watching the auction (if logged in as a buyer)

```
SELECT COUNT(buyerID)
FROM Watching
WHERE auctionID = '$auction_id' AND buyerID = '$user_id'
```

- Generates a list of 5 recommendations for insertion into a carousel at the bottom of the page for the user viewing it (doesn't need to be logged in). The query itself returns a table with two columns:
    1. A list of active auctions (excluding the current auction) which watchers of this auction (excluding the current user if logged in as a bidder) are also watching
    2. A count of the number of buyers watching that auction (its 'score'). The logic of the recommendation system is based on watching activity rather than bids, since users often watch before bidding, and watch several of similar kinds of products before deciding which one (often only one) to bid on.

```
SELECT auctionID, COUNT(buyerID) AS score
FROM Watching AS w
WHERE buyerID IN ($query_buyers)
AND auctionID != '$this_auction_id'
AND '$current_time' > (SELECT startDate
                       FROM Auctions
                       WHERE auctionID = w.auctionID)
AND '$current_time' < (SELECT endDate
                       FROM Auctions
                       WHERE auctionID = w.auctionID)
GROUP BY auctionID
ORDER BY score DESC
LIMIT 5
```

where the below query is stored as a string (not called separately):

- Returns a list of buyers (or, if logged in as a buyer, other buyers) watching this auction

```
$query_buyers = SELECT buyerID
                FROM Watching
                WHERE auctionID = '$this_auction_id'
                AND buyerID != '$buyer_id'
```

- Retrieves basic information on the recommended auctions

```
SELECT title, endDate, sellerID, categoryID
FROM Auctions
WHERE auctionID = '$auctionID'
```

- Retrieves the recommended auctions' categories

```
SELECT categoryName FROM Categories WHERE categoryID = '$categoryID'
```

- Retrieves the recommended auctions' sellers' usernames

```
SELECT username FROM Sellers WHERE sellerID = '$sellerID'
```

- Retrieves the current price of the recommended auctions (if null, the script sets the current price to the start price collected earlier)

```
SELECT MAX(bidAmount) AS maxBid FROM Bids WHERE auctionID = '$auctionID'
```

php/check_bid.php

- Run by an AJAX call in listing.php to check the bid is a valid amount

```
SELECT bid_check('$username')
```

place_bid.php

- Retrieves the ID of the highest bidder before a new bid is placed, the auction title to include in the email notification, and the end date of the auction

```
SELECT b.buyerID, a.title, a.endDate
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
WHERE a.auctionID = '$auction_id'
AND b.bidAmount = (SELECT MAX(bidAmount)
                   FROM Bids
                   WHERE auctionID = '$auction_id')
```

- Inserts the bid into the database

```
INSERT INTO Bids (bidDate, bidAmount, buyerID, auctionID)
VALUES ('$bid_date', '$bid_amount', '$buyer_id', '$auction_id')
```

- Retrieves a list of buyers who have this auction in their watchlist

```
SELECT w.buyerID, b.username, b.email
FROM Watching w
JOIN Buyers b ON b.buyerID = w.buyerID
WHERE auctionID = '$auction_id'
```

- If the bid was successfully placed, check if the bidder has the auction on their watchlist already (if not, they are prompted to watch it to receive notifications)

```
SELECT COUNT(buyerID)
FROM Watching
WHERE auctionID = '$auction_id' AND buyerID = '$buyer_id'
```

mybids.php

- Retrieving the categories from the database to filter auctions by

```
SELECT categoryName, categoryID FROM Categories
```

- The first part of the main query: collects the information to display in the auction cards, except the user's bid history

```
$sql_1 =
SELECT a.*
FROM ($sql_temp) AS a
LEFT JOIN ($sql_temp2) AS b ON a.auctionID = b.auctionID
WHERE 0=0
```

To this, simple `WHERE` *conditions are concatenated before the query is called based on the filters chosen by the user. For example, if the user filters for auctions in the category 'Electronics' (ID = 3), which the user has won, ordered by the sale price (descending), the code will append to the* `WHERE` *clause:*

```
AND categoryID IN ('3')
AND '$currentTime' > a.endDate
AND (a.YourHighestBid = b.MaxBid)
ORDER BY maxBid DESC
```

Full details can be found in lines 217-261 of the file.

Two simpler queries are stored as strings to assist in building the main query. Only one query, the main query, is called:

- Retrieves all relevant auction information, including the current user's highest bid (might not be the auction's current maximum bid)

```
$sql_temp =
SELECT a.auctionID, s.username, a.title, a.endDate, c.categoryID,
       c.categoryName, MAX(b.bidAmount) AS YourHighestBid
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
JOIN Categories c ON a.categoryID = c.categoryID
JOIN Buyers b2 ON b2.buyerID = b.buyerID
JOIN Sellers s ON s.sellerID = a.sellerID
WHERE b.buyerID = '$buyer_id' AND b2.username = '$username'
AND a.auctionID IN (SELECT DISTINCT a.auctionID
                    FROM Auctions a
                    JOIN Bids b ON a.auctionID = b.auctionID
                    WHERE b.buyerID = '$buyer_id')
GROUP BY a.auctionID, s.username, a.title, a.endDate, c.categoryName
```

- Retrieves the maximum bid for each auction

```
$sql_temp2 =
SELECT a.auctionID, MAX(b.bidAmount) AS MaxBid
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
GROUP BY a.auctionID
```

- Determines the current state of the auction, including the username of the current highest bidder

```
SELECT a.auctionID, b.bidAmount, a.reservePrice, b.buyerID, bu.username
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
JOIN Buyers bu ON b.buyerID = bu.buyerID
WHERE a.auctionID = '$productID'
AND b.bidAmount = (SELECT MAX(bidAmount)
                   FROM Bids
                   WHERE auctionID = '$productID')
```

- Retrieves the current user's bid history for the auction (for insertion into an expandable table in the auction card)

```
SELECT bidDate, username, bidAmount
FROM Bids, Buyers
WHERE Bids.buyerID = Buyers.buyerID
AND auctionID = '$productID'
AND Bids.buyerID = '$buyer_id'
ORDER BY bidDate
```

php/check_auction_end.php

- Retrieves the auctions that have ended in the last 10 minutes and relevant information about them (for the scheduled email notification task)

```
SELECT a.auctionID, a.title, a.reservePrice, s.username AS sellerUsername,
       s.email AS sellerEmail, b1.maxBid, CASE WHEN a.reservePrice >
       IFNULL(b1.MaxBid, 0) THEN 0 ELSE 1 END AS sold, b2.username AS
       buyerUsername, b2.email AS buyerEmail
FROM Auctions a
LEFT JOIN Sellers s ON s.sellerID = a.sellerID
LEFT JOIN ($sql_0) b1 ON b1.auctionID = a.auctionID
LEFT JOIN Buyers b2 ON b1.buyerID = b2.buyerID
WHERE a.endDate < NOW() AND TIMESTAMPDIFF(MINUTE, a.endDate, NOW()) <= 10
```

where the below query is stored as a string (not called separately):

- Retrieves the maximum bid and the buyer who made that bid for each auction

```
$sql_0 =
SELECT BidsTemp.auctionID, Bids.buyerID, BidsTemp.maxBid
FROM Bids
JOIN (SELECT auctionID, MAX(bidAmount) AS maxBid
      FROM Bids
      GROUP BY auctionID) BidsTemp ON Bids.bidAmount = BidsTemp.maxBid
                                   AND Bids.auctionID = BidsTemp.auctionID
```

5.5 Capability 5

watchlist_funcs.php

- Adding an auction to a buyer's watchlist

```
INSERT INTO Watching (auctionID, buyerID) VALUES ('$auction_id', '$buyer_id')
```

- Removing an auction from a buyer's watchlist

```
DELETE FROM Watching WHERE auctionID = '$auction_id' AND buyerID = '$buyer_id'
```

mywatchlist.php

- Retrieving the categories from the database to filter auctions by

```
SELECT categoryName, categoryID FROM Categories
```

- The first part of the main query: collects the information for the auction cards

```
$sql_1 =
SELECT a.*
FROM ($sql_temp) AS a
LEFT JOIN ($sql_temp2) AS b ON a.auctionID = b.auctionID
WHERE 0=0
```

To this, simple `WHERE` *conditions are concatenated before the query is called based on the filters chosen by the user. For example, if the user filters for active auctions in the category 'Electronics' (ID = 3), which the user is bidding on but not currently winning, ordered by how soon they are ending, the code will append to the* `WHERE` *clause:*

```
AND categoryID IN ('3')
AND '$currentTime' < a.endDate
AND (IFNULL(b.YourHighestBid, 0) < IFNULL(a.MaxBid, 0) OR a.MaxBid IS NULL)
ORDER BY endDate
```

Full details can be found in lines 212-256 of the file.

Two simpler queries are stored as strings to assist in building the main query. Only one query, the main query, is called:

- Retrieves all relevant auction information, including the auction's current highest bid (might not be the current user's own)

```
$sql_temp =
SELECT w.auctionID, a.title, a.descript, c.categoryID, c.categoryName,
       a.endDate, a.startPrice, a.reservePrice, s.username,
       MAX(b.bidAmount) AS MaxBid
FROM Watching w
JOIN Auctions a ON w.auctionID = a.auctionID
LEFT JOIN Categories c ON a.categoryID = c.categoryID
LEFT JOIN Sellers s ON a.sellerID = s.sellerID
LEFT JOIN Bids b ON w.auctionID = b.auctionID
WHERE w.buyerID = '$buyer_id'
GROUP BY w.auctionID, a.title, a.descript, c.categoryID, c.categoryName,
         a.endDate, a.startPrice, a.reservePrice, s.username
```

- Retrieves the user's maximum bid for each auction they are bidding on

```
$sql_temp2 =
SELECT a.auctionID, MAX(b.bidAmount) AS YourHighestBid
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
WHERE b.buyerID = '$buyer_id'
GROUP BY a.auctionID
```

- Determines the current state of the auction, including the username of the current highest bidder

```
SELECT a.auctionID, b.bidAmount, a.reservePrice, b.buyerID, bu.username
FROM Auctions a
JOIN Bids b ON a.auctionID = b.auctionID
JOIN Buyers bu ON b.buyerID = bu.buyerID
WHERE a.auctionID = '$productID'
AND b.bidAmount = (SELECT MAX(bidAmount)
                   FROM Bids
                   WHERE auctionID = '$productID')
```

5.6 Capability 6

recommendations.php

- Generates an ordered list of 10 recommendations for the user by 'score'. The query itself returns a table with two columns:
    1. Active auctions which the current user hasn't bid on before but the 'similar' bidders from the query below *have* bid on
    2. A count of the associated similarity scores for each auction: increasing in both 1. number of bids received & 2. the 'similarity' of its bidders. An auction with a lot of slightly similar bidders might be more interesting than an auction with only a few highly similar bidders (from the business' perspective).

```
SELECT auctionID, COUNT(similarity) AS score
FROM Bids AS b, ($query_other_bidders) AS q
WHERE b.buyerID = q.buyerID
AND b.auctionID NOT IN ($this_buyer_auctions)
AND '$current_time' > (SELECT startDate
                       FROM Auctions
                       WHERE auctionID = b.auctionID)
AND '$current_time' < (SELECT endDate
                       FROM Auctions
                       WHERE auctionID = b.auctionID)
GROUP BY auctionID
ORDER BY score DESC
LIMIT 10
```

where the two below queries are stored as strings (not called separately):

- Auctions this buyer has bid on before

```
$this_buyer_auctions = SELECT DISTINCT auctionID
                       FROM Bids
                       WHERE buyerID = '$this_buyer_ID'
```

- Returns a table with two columns (ignores buyerID '1' as that is the default value):
    1. Bidders who have bid on at least 1 auction that the current user has also bid on
    2. 'similarity' covariate (how many auctions that bidder has bid on that the current user has also bid on)

```
$query_other_bidders
= SELECT buyerID, COUNT(DISTINCT auctionID) AS similarity
  FROM Bids
  WHERE buyerID NOT IN ($this_buyer_ID, '1')
  AND auctionID IN ($this_buyer_auctions)
  GROUP BY buyerID
```

- Retrieves basic information on the recommended auctions

```
SELECT title, endDate, sellerID, categoryID
FROM Auctions
WHERE auctionID = '$auctionID'
```

- Retrieves the recommended auctions' categories

```
SELECT categoryName FROM Categories WHERE categoryID = '$categoryID'
```

- Retrieves the recommended auctions' sellers' usernames

```
SELECT username FROM Sellers WHERE sellerID = '$sellerID'
```

- Retrieves the current price of the recommended auctions (must have been at least due to the process they are selected using)

```
SELECT MAX(bidAmount) AS maxBid FROM Bids WHERE auctionID = '$auctionID'
```