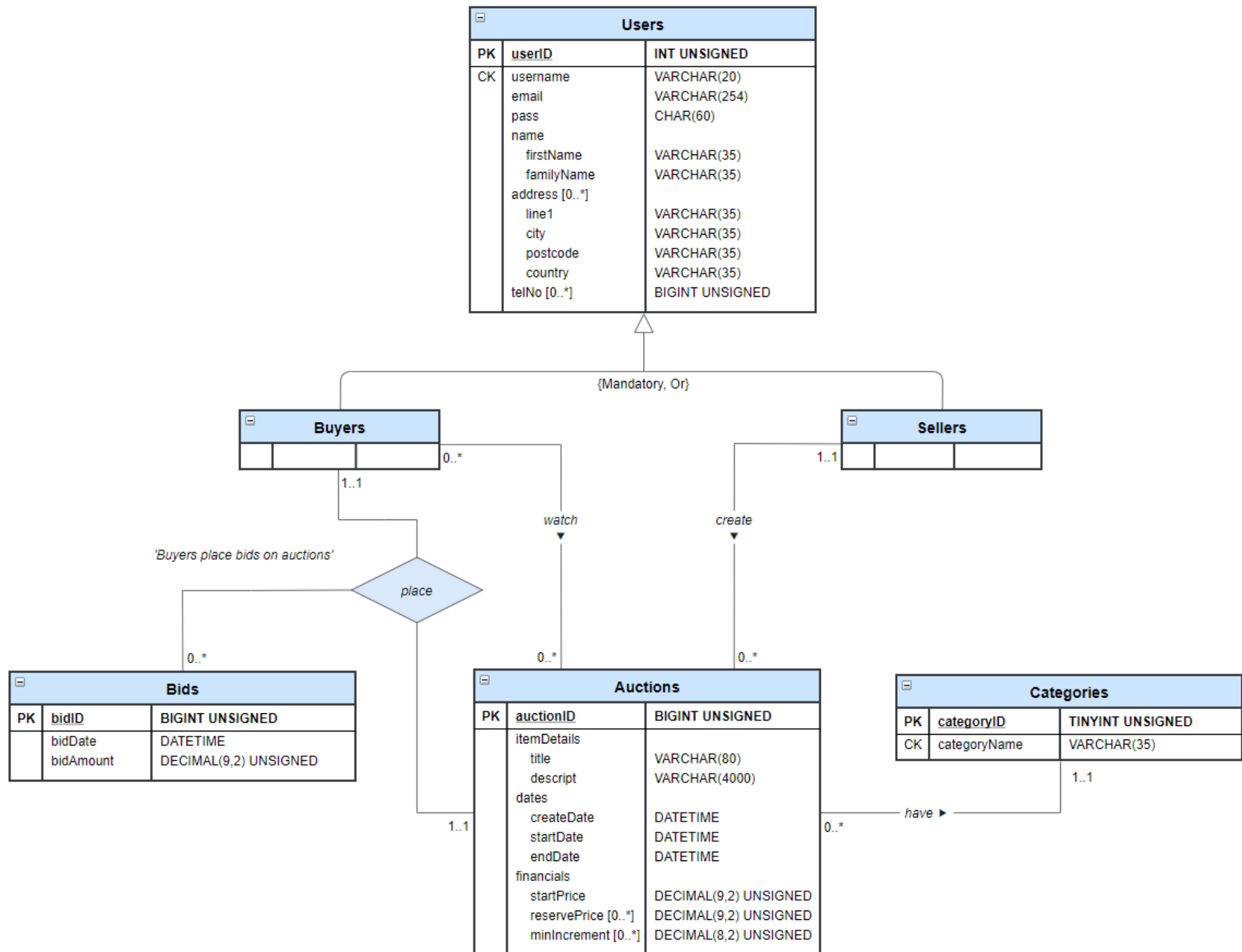


Database design wireframe

1 – Conceptual design (building conceptual data model)

- 1.1 - *Identify entity types*
 - Data dictionary documenting entities (or 'entity dictionary')
 - Highlight requirements table
 - At this stage don't have address, telNo or countries as separate entities
- 1.2 - *Identify relationship types*
 - List of entities and relationships e.g. "Sellers create auctions"
 - Highlight requirements table
 - First-cut ER diagram without attributes incl. multiplicity
 - At this point assuming auctions can only have one category (not a big difference in ER diagram, but would make a big difference in the schema); also assuming only one 'parent' level of categories – no sub-categories
 - Check for fan and chasm traps
 - Augmented data dictionary (or 'relationship dictionary')
- 1.3 - *Identify and associate attributes with entity or relationship types in ERD*
 - Discuss logic about attributes chosen:
 - Huge number of possible attributes and associated functionality for an auction site. Focus on those:
 - Necessary for stated functionality (e.g. reservePrice)
 - Simple to include in a complete way (e.g. minIncrement)
 - Necessary in general (e.g. addresses for sellers to post to, and for payment assuming use own payment platform, incl. both billing & shipping addresses.
 - Identify entity and relationship attributes (highlight requirements table)
 - Simple/composite
 - Single/multi-valued
 - Derived. Note here about derived attributes – many are 'natural' and technically not going to break 3NF (later), but generally not a good idea unless clear evidence of a performance advantage since they increase complexity
 - Check only associated with one entity
 - Augmented data dictionary (or 'attribute dictionary')
- 1.4 - *Determine attribute domains*
 - Define domains of each attribute
 - Allowable set of values (e.g. M or F)
 - Sizes and formats (e.g. 1 character)
 - Assuming this platform is designed for scale rather than testing, so allowing lots of users, bids, etc. from the beginning

- Check phone number, email address and passwords for formatting/requirements before inserting into the database. Phone number stored as DATETIME rather than TIMESTAMP.
 - Possibly better for buyerID/sellerID to be different in format e.g. (Bxx) (Yxx). Not crucial so not implemented here.
- Add to 'attribute dictionary'
- **1.5 - Determine candidate, primary, and alternate key attributes**
 - Identify CKs: minimal set of attributes uniquely identifying occurrence of the entity
 - PK: chosen for identification (PPK if composite)
 - Include userID so users can change username easily
 - Include auctionID since allow duplicate titles (need a CK per table) – but note, technically a 'weak' entity
 - Include categoryID so category name can be changed (e.g. to superset); possibly shorter
 - Include bidID so have a shorter identifier (than incl. buyers, auctions) – but note, technically a 'weak' entity
 - AK: other CKs (NB: allow titles to be non-unique)
 - username
 - categoryName
 - Select based on size, probability of values being changed, shortest length or easiest to use
 - Add to data dictionary
 - At this stage, also identify strong and weak entities
 - *Strong*: can assign a primary key to the entity
 - *Weak*: cannot identify a primary key to the entity. PK of weak entity can be identified only once we've mapped the weak entity and its relationship with its 'owner' entity to a relation through placement of a foreign key in the relation.
- **1.6 - Consider use of enhanced modelling concepts (optional step)**
 - Specialisation/generalization (specific instance of the parent). Done here for buyers & sellers
 - Aggregation (child can exist independently of parent)
 - Composition (child cannot exist independently of parent)
- **1.7 - Check model for redundancy**
 - Remove redundant entities
 - Re-examine 1:1 relationships (i.e. don't have two entities representing same object)
 - Remove redundant relationships
 - i.e. don't have relationship where same information can be gained via other relationships, as developing a minimal data model.
- **1.8 - Validate conceptual data model against user transactions**
 - Some examples of 'written' transactions (like SQL queries) and show how they are possible
- **1.9 - Review conceptual data model with user**
 - Received feedback on week 3 draft of ER diagram in week 4



2 – Logical design (building logical data model)

- 2.1 - Derive relations for logical data model

- Superclass/subclass
 - {Mandatory, Or} so create one relation each for buyers and sellers, no Users relation
- 1NF:
 - Flatten composite attributes
 - No nulls – if optional, separate entity
 - Address, telNo need separate entities
 - Include addressID as PK to reduce data duplication, possibly shorter. No need for telNo as only one attribute in entity.
 - Likewise, can include a country entity since this is a fixed list (and can be used on the front-end). Have a countryID PK to reduce data duplication (shorter) and in case countries change names (does happen occasionally)
 - For addresses and telNos, treat these as a 1:* rather than *:.* relationship – see *notes*. Therefore need separate 'buyer' and 'seller' entities for both of these (since FK can't reference two entities at once).
 - reservePrice, minIncrement can be handled on the DB and PHP side by setting these to 'startPrice' and '0.01' respectively if null, so no separate entities needed
- Other strong entities (create a relation + flatten composite attributes)
 - Categories
- Weak entities (create a relation + flatten composite attributes; PK already identified)
 - Bids
 - Auctions
- 1:* relationships ("1" side is designated parent and "*" side gets PK of parent as an FK)
 - Users (buyers/sellers) (1) have addresses (*) and telephone numbers (*)
 - Addresses (buyer/seller) (*) have countries (1)
 - Sellers (1) *create* auctions (*)
 - Auctions (*) *have* categories (1)
- *:.* relationships (create a relation to represent the relationship incl. PKs of both entities participating as FKs and any other relevant attributes; one or both of FKs acts as PK)
 - Buyers (*) *watch* auctions (*)
- Complex relationship types (create a relation representing the relationship incl. attributes part of the relationship, favouring PK of entity with 'many' cardinality near it)
 - Buyers (1) place bids (*) on auctions (1)
 - NB: This will just create a dedicated table as an extension of the bids table with FKs from Buyers and Auctions
 - Could replicate ER diagram as a visual 'schema', with added FKs and an extra 'Watching' table attached to 'watch' relationship (labelled 'Watching')

- For 'Watching' table, don't add a new PK as not referenced elsewhere, so no advantage in terms of storage saving
- *2.2 - Validate relations using normalization*
 - Check step-by-step that there are no FDs violating 2NF or 3NF
 - Done – discussion around addresses needed (e.g. for the UK)
 - 4x4 for pairwise checking of quads, triples and doubles etc. for UK; if not true for UK then not true in general
 - Discussion around 'over normalization' and intention
 - Discuss 'null' values and 1NF
 - Can systematically do this as in the book for each relation
- *2.3 - Validate relations against user transactions*
 - Check actually works for given request manually. Implicitly done this when testing the front-end.
- *2.4 - Check integrity constraints*
 - Required data – all data (no nulls allowed)
 - Attribute domain constraints – already done earlier
 - Multiplicity – already done earlier;
 - Multiplicity constraints in general reflect understanding of requirements, not a wider range of what could be logically possible (e.g. users can stop an address from being associated from their account, which would imply a 0..* multiplicity, or user accounts can be deleted)
 - Entity integrity – PKs identified, CKs (unique) identified
 - Referential integrity
 - Are FK nulls allowed? No, since no nulls allowed.
 - How to ensure referential integrity? 2x3 of child, parent vs. insert, delete, update – 3 relevant DML functions; only interested in 4 cases (defined after schema listed later in document)
- *2.5 - Review logical data model with user*
 - Not done as this is an assessment
- *2.6 - Merge logical data models into global model (optional step)*
 - Skipped as only creating a single user view, but in reality might want to create multiple views for different roles e.g. admin vs. data scientists especially wrt. the creation of the 'Defaults' table
- *2.7 - Check for future growth*
 - Discussion about extensibility – list of additional features e.g. email verifications, phone verifications, 2FA, primary/secondary/home/mobile, different email and phone number types, images, ratings and how difficult it would be to include

Schema:

Buyers (buyerID, username, email, pass, firstName, familyName)

BuyerAddresses (addressID, line1, city, postcode, *countryID*, *buyerID*)

BuyerTels (telNo, *buyerID*)

Sellers (sellerID, username, email, pass, firstName, familyName)

SellerAddresses (addressID, line1, city, postcode, *countryID*, *sellerID*)

SellerTels (telNo, *sellerID*)

Countries (countryID, countryName)

Bids (bidID, bidDate, bidAmount, *buyerID*, *auctionID*)

Watching (*auctionID*, *buyerID*)

Auctions (auctionID, title, descript, createDate, startDate, endDate, startPrice, reservePrice, minIncrement, *sellerID*, *categoryID*)

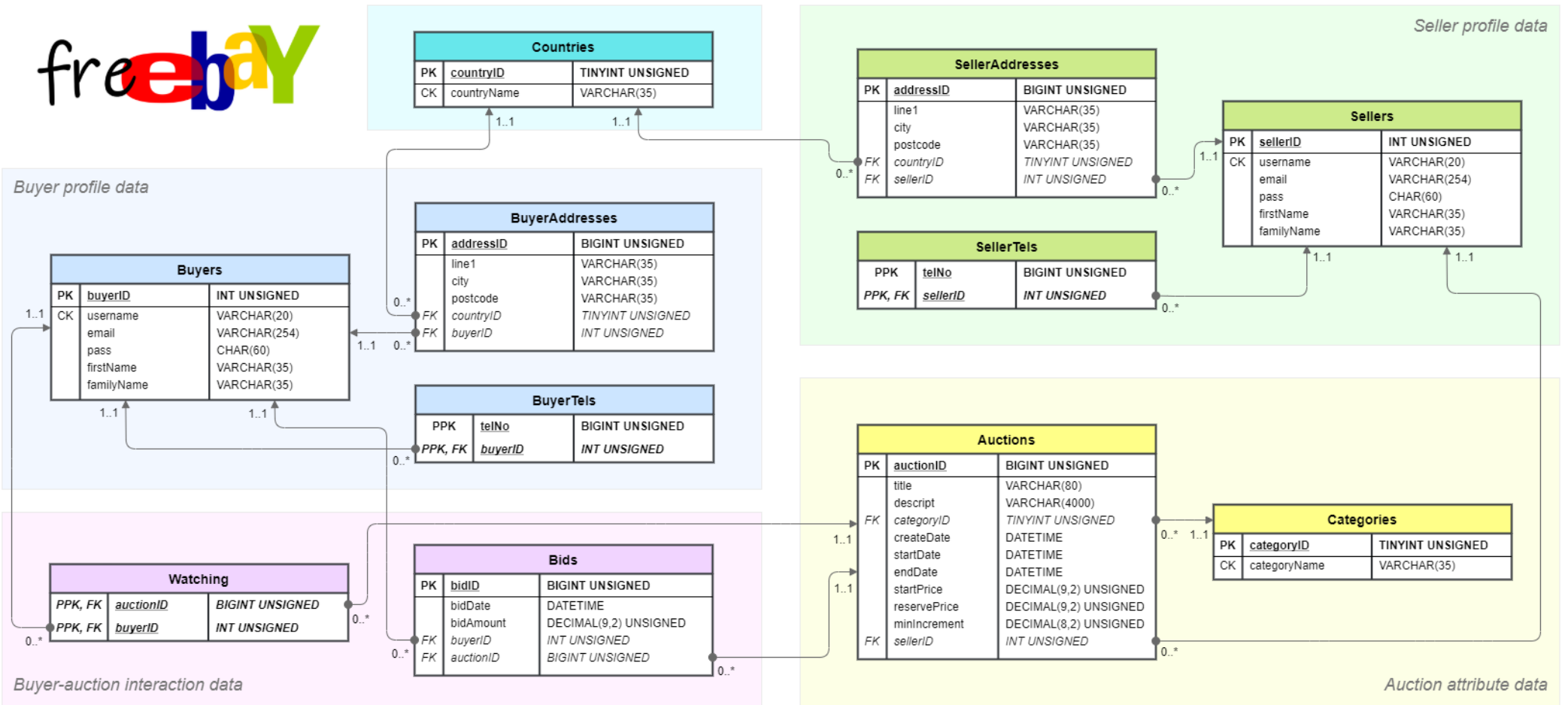
Categories (categoryID, categoryName)

- **Referential integrity:**

- On insertion of new record to child relation, check FK value equal to a value of parent tuple
 - Done automatically in MySQL with InnoDB
- On update of child FK value, check equal to a value of existing parent tuple (done automatically in MySQL with InnoDB)
 - Done automatically in MySQL with InnoDB
- On update to parent PK value, cascade update to child entities' FK values
 - Need to code as different options here. We want update (though the PK is highly unlikely to change)
- For deletion, need to think about 1) if want to allow (simpler to just say no for all) and 2) how
- Allow deletion for all FK parents (except country, as countries are unlikely to change – in event of a country ceasing to exist, can change name to new name and select 'unique' when pulling the list on PHP) even though not included as a feature on UI side currently since may want to delete things directly e.g. inactive accounts, old auctions, redundant categories. On deletion from parent tuple.
 - *** Note: InnoDB does not support ON DELETE, SET DEFAULT, so when it says that below it is actually achieved differently in the code (a combination of a trigger and 'ON DELETE, CASCADE' ***
 - **BuyerAddresses** *buyerID*: cascade
 - If a buyer is deleted, then no need for address information (since assuming an address can only be held by one buyer)
 - **SellerAddresses** *sellerID*: cascade
 - If a seller is deleted, then no need for address information (since assuming an address can only be held by one seller)
 - **BuyerTels** *buyerID*: cascade

- If a buyer is deleted, then no need for telNo information (since assuming a telNo can only be held by one buyer)
- **SellerTels** *sellerID*: cascade
 - If a seller is deleted, then no need for telNo information (since assuming a telNo can only be held by one seller)
- **BuyerAddresses** *countryID*: no action
 - Most likely a mistake, other ways to handle this
- **SellerAddresses** *countryID*: no action
 - Most likely a mistake, other ways to handle this
- **Bids** *buyerID*: set default
 - Mechanism to avoid allowing nulls but keep bid information for completed listing searches. More important to avoid nulls on insertion, since that should always be linked to a buyer (and also for 1NF).
 - *More complex version would only allow deletion if bids are for non-active auctions*
- **Bids** *auctionID*: cascade
 - *If an auction is deleted, then no need for bid information*
- **Watching** *buyerID* : cascade
 - If a buyer is deleted whilst watching an auction, no need for watching information
- **Watching** *auctionID*: cascade
 - If an auction is deleted whilst someone is watching it, no need for watching information
- **Auctions** *sellerID*: set default
 - Mechanism to avoid allowing nulls but keep auction information for completed listing searchers. More important to avoid nulls on insertion, since that should always be linked to a seller (and also for 1NF).
 - *More complex version would only allow deletion if auctions are non-active*
- **Auctions** *categoryID*: set default
 - *Allow auctions to keep a category (i.e. 'Other'): better to update rather than delete categories but might lead to a strange organisation of categories*
 - But, if not happening in isolation (e.g. also adding new categories), will need to be careful in update
- **Default values:**
 - **Bids** *buyerID*: 1 (N/A)
 - **Auctions** *sellerID*: 1 (N/A)
 - **Auctions** *categoryID*: 1 (Other)

- To prevent deletion **create an additional 'Defaults' table** which only top-level administrators can access – other normal 'admins' cannot see it as they aren't granted rights on it – that references the above tables with a foreign key constraint. Since most 'action' will relatively be in the bids, sellers and categories tables, this helps to ensure integrity.
 - The below database scheme diagram is therefore technically missing the 'defaults' table, but omitted for simplicity since not a core part of scheme except maintaining integrity (i.e. contains no information relevant to actual entities and relationships)
 - In a production implementation, might have a portal for DB administrators to interact with DB with its own logic to prevent deletion of certain rows (i.e. only certain records are visible)



3 – Physical design (translating logical data model for target DBMS)

- Has been done simultaneously (in part) with the above as know capabilities of MySQL
- Base relations in current form can all be represented in MySQL fine
- Discussion of design representation of **derived data**:
 - Doesn't technically break 3NF for parents to include summarised data from child entities, but goes against spirit of normalisation by introducing redundancy and complexity
 - Might give performance advantages, especially for those which are accessed more than they are updated (e.g. numBids), and performance is critical in eCommerce applications
 - Reduce complexity and query in real-time when needed, but keep this as an option if performance is unsatisfactory – very important for eCommerce sites
- **General constraints**: additional constraints to ensure data integrity:
 - Usernames must be unique across both buyers and sellers (so don't have 'UNIQUE' constraint in table creation)
 - Implemented using triggers (one each for Buyers and Sellers)
 - Start date must be after create date
 - Implemented at table creation
 - End date must be later than start date
 - Implemented at table creation
 - Reserve price must be greater than start price
 - Implemented at table creation
 - minIncrement must be greater than 0
 - Implemented at table creation
 - Bids should be greater than current max bid + minIncrement
 - Implemented using a stored procedure and a trigger
- *And lots more possible...*
 - Don't delete seller account if have active auctions
 - *For current implementation, not allowing buyers to delete accounts*
 - *Could be added later with a trigger/stored procedure*
 - Don't delete buyer account if currently a leading bidder on an auction
 - *For current implementation, not allowing sellers to delete accounts*
 - *Could be added later with a trigger/stored procedure*
 - Etc.