eMarcedo — an *eBay* clone web app

Guntakanti Sai Koushik — 180050035
Potta Nayan Akarsh — 180050074
Mekala Anmol Reddy — 180050059
Mavuri Siva Krishna Manohar — 180050057

Relational Schema (in next page):

- This schema is in BCNF, there are no internal dependencies, except *email* being a unique attribute in *person* table.
- Since is it also a candidate key, *person* is in BCNF.
- No de-normalization has been done.
- We aren't using any triggers because we'd already manually written the updates on changing what would have been triggered in the SQL for the use cases.

Constraints

- **person relation:**
- email is not null, unique
- ($amount\_on\_hold <= balance$)
- **auction_item relation:**
- seller_id is not null
- status='open' or status='closed' or status='auctioned' or status='shipping' or status='shipped' or status='out-for-delivery' or status='delivered'
- ($quantity > 0$)
- ($delivery\_factor >= 0$)
- (best_bid is null or $best\_bid >= price$)
- ($close\_time > start\_time$)
- foreign key(best_bidder) references person
- foreign key(seller_id) references person
- **direct_sale_item relation:**
- seller_id is not null,
- status='open' or status='closed' or status='auctioned' or status='shipping' or status='shipped' or status='out-for-delivery' or status='delivered'
- ($quantity > 0$)
- ($delivery\_factor >= 0$))
- foreign key(seller_id) references person

- foreign key(buyer_id) references person
- **auction_item_tags relation:**
- foreign key(identifier) references auction_item
- **direct_sale_item_tags relation:**
- foreign key(identifier) references direct_sale_item
- **bid relation:**
- foreign key(aitem_id) references auction_item
- foreign key(person_id) references person

DB indexes

- **Index on *seller_id* in auction_item relation**
  In many use cases like fetching all the auctions put up by the user, updating the balance of the seller after sale, the query is conditioned on the seller_id of auction_item. So, creating an index on it would improve the efficiency on those queries.
- **Index on *seller_id* in direct_sale_item relation**
  For the same reason as above.
- **Index on *buyer_id* in direct_sale_item relation**
  In use cases where we are finding out the items bought by a particular user, the query is conditioned on buyer_id and hence indexing on it would likely improve efficiency.
- **Index on *tag* in auction_item_tags relation**
  In use case(s) where the items are searched upon by the tags, the search would be efficient with the help of indexing on tag.
- **Index on *tag* in direct_sale_item_tags relation**
  For the same reason as above.

Note: Since we will have indexes on primary keys by default, email since unique condition is imposed on it and location indexing is taken care of by postgis, no special indexes need to be defined on them.

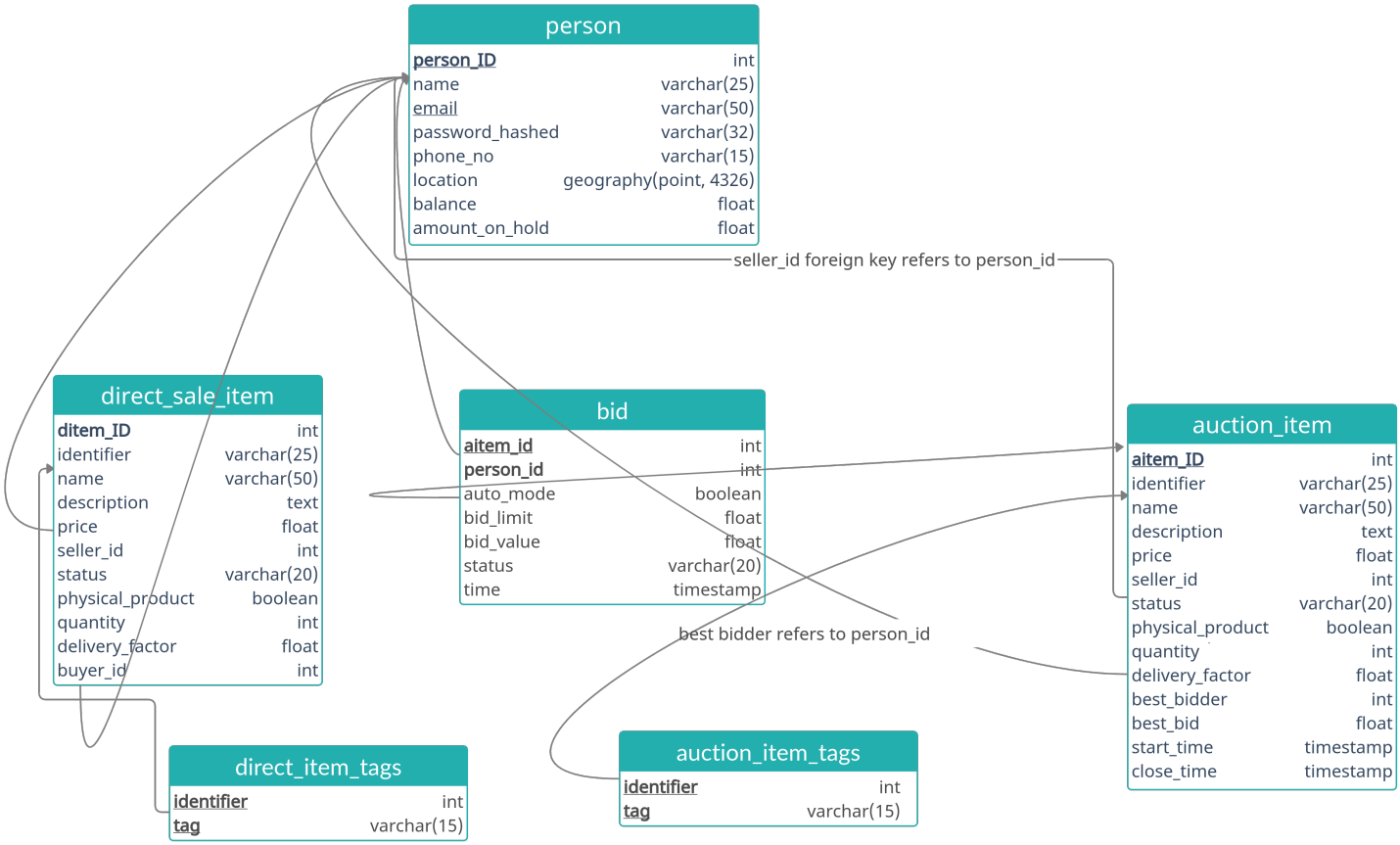

Figure 1: eMarcedo Schema

Data Generation and loading

We have used subsets of the following downloaded data sets:

- https://www.kaggle.com/kaggle/us-baby-names
  This data set contains millions of names along with some other information. We used a subset of it (the first 10000 entries with unique names) to generate the person data.
- https://data.world/promptcloud/product-details-on-flipkart-com
  This data set contains information about 20000 products being sold on flipcart with useful attributes like name, description, price, category tree, etc. We filtered out data that doesn't have information about price and some other data with too lengthy names. Then, we used it to construct data for the auction_item, direct_sale_item and used the category tree to construct auction_item_tags and direct_sale_item_tags.

For the attributes unavailable from the data set downloaded, we have used randomisation to get relevant values in some cases like seller_id, quantity, delivery factor, phone_no and relevant seed values in other cases.

The helper file *datagen.py* used for generation of our data from the above mentioned datasets is attached in the appendix for reference along with the instructions on directory structure while execution. The data load script *dataload.sql* is also attached there which loads the seed data generated by *datagen.py* into the database.

Software Webpages in `html`, `NodeJS` backend, `ReactJS` frontend, `PostgreSQL` (with `Postgis` for geo-spatial data). Our controller is externalised – built using Node.js

Use Cases

- **Non-User**

1. **Account for a user**

Inputs	name, email-id, password, re-enter password, phone number, location, balance; check for duplicate email
Main success path	Creates a user account with the entered details
Exceptions	Verify password and re-entered password are same, if not refresh with other entered details
Post conditions	Person gets signed up, logged in, gets added into database and goes to home page. Redirect back here for unmatched passwords or repeated emails.
SQL	`INSERT INTO person(name, email, password_hashed, phone_no, location, balance, amount_on_hold) VALUES(uname, email-id, pass_hash, phno, given_loc, balance, 0.0);`

- **User**

1. **Logging in credentials**

Inputs	email, password;
Main success path	Successfully logged in and we land on the home page for users.
Exceptions	Incorrect password message is displayed, redirect to same page, to enter the correct password
Post conditions	No change to database, after authentication go to home page of user ("get home details" use case)
SQL	`SELECT password_hashed FROM person WHERE email=email_id;`

2. **Log out**

Inputs	email, password;
Main success path	Already logged in, now logged out and we land on the sign-in page.
Post conditions	User logged out

3. **Enter home**

Inputs	many use cases redirect here, clicking the logo button on top too
Main success path	All current bids, sales, past buys, bids, sales and associated receipts displayed (as drop-down) and button to go into them further and get details. Buttons to upload sale, update, a search bar balance
SQL	`SELECT * FROM bid NATURAL JOIN auction_item NATURAL JOIN auction_item_tags WHERE person_id=curr_uid; SELECT * from auction_item NATURAL JOIN auction_item_tags WHERE seller_id=curr_uid; SELECT * from direct_sale_item NATURAL JOIN direct_sale_item_tags WHERE seller_id=curr_uid;`

4. **Adding Balance**

Inputs	Credit Value (positive integer)
Main success path	Successfully credited the entered amount, and update balance, redirect to same home page
Post conditions	balance is updated in the database
SQL	`UPDATE person SET balance = credit + balance WHERE email = email_id;`

5. **Upload Auction**

Inputs	product_id, name, description, price, quantity, phys_prod, quantity, delivery_factor, tags
Main success path	Item is up for sale and available on searching by other users, seller redirected to his home page, where new item is displayed
Post conditions	Item is added to the database
SQL	`INSERT INTO auction_item(identifier, name, description, price, seller_id, status, physical_product, quantity, delivery_factor, best_bidder, best_bid, start_time, close_time) VALUES(pid, name, descr, price, curr_uid, "open", phy_prod, qnty, d_fact, NULL, NULL, NOW(), NULL);` and for each *tag* given in input do: `INSERT INTO auction_item_tags values(identifier, tag);`

6. **Upload Direct Sale**

Inputs	product_id, name, description, price, quantity, physical, quantity, delivery_factor, tags;
Main success path	Item is up for sale and available on searching by other users, seller redirected to his home page, where new item is displayed
Post conditions	Item is added to the database
SQL	`INSERT INTO direct_sale_item(identifier, name, description, price, seller_id, status, physical_product, quantity, delivery_factor, buyer_id) VALUES(product_id, name, description, price, curr_user_id, "open", phy_prod, qnty, d_fact, NULL);` and for each *tag* given in input add the entry by: `INSERT INTO direct_sale_item_tags values(identifier, tag);`

7. **Click on a sale or bid**

Inputs	Click on a sale in the in the home page or in the search results
Main success path	Redirected to page showing complete description of product and option to remove sale, update sale details etc.
Post conditions	No change to database
SQL	`SELECT * FROM auction_item WHERE aitem_id=sale_id` or `SELECT * FROM direct_sale_item WHERE ditem_id=sale_id;` along with info about bid if user had bid for the item `SELECT * FROM bid WHERE aitem_id=sale_id and person_id=curr_uid;`

8. **Delete a sale**

Inputs	select delete on the product description page
Main success path	Product deleted from auction, triggering a bid delete, then bidders notified and their on hold balances are updated
Post conditions	product does not appear anywhere anymore, bidder gets message with details of removal on home page
SQL	`WITH person_balance_item(person_id, amount_on_hold, aitem_id) AS (SELECT person_id, amount_on_hold, aitem_id FROM person NATURAL JOIN bid NATURAL JOIN auction_item WHERE aitem_id = sale_id) UPDATE person SET amount_on_hold = amount_on_hold - (select P.bid_value FROM person_balance_item P WHERE P.person_id=person_id) WHERE person_id IN (SELECT person_id FROM person_balance_item);` and then `DELETE FROM auction_item WHERE aitem_id=sale_id` or, for direct item just delete the sale by: `DELETE FROM direct_sale_item WHERE aitem_id=sale_id;` (can be changed to a more efficient query by updating each bidder individually by making a separate query for each instead of doing all within a single query)

9. **Search**

Inputs	item name, search tags etc entered in search bar of homepage
Preconditions	User is in the search page
Main success path	redirect to search results page, both auctioned and direct sale items
Post conditions	
SQL	`SELECT aitem_id, identifier, name, quantity FROM auction_item;` `SELECT ditem_id, identifier, name, quantity FROM direct_sale_item;` the usage of these commands (like searching based on tag, similarity, ordering based on relevance) will depend upon how search is implemented – this is not final

10. **Place a direct order**

Inputs	On the page of product details place an order
Main success path	Order is placed successfully if balance available and product is not owned by oneself, on-hold balances updated
Post conditions	Status of item is updated
SQL	`UPDATE direct_sale_item SET status = "sold", buyer_id = curr_uid;`

11. **Bid**

Inputs	bid value, bid mode, limit;
Main success path	Bid is placed if balance available, best bid and associated on-hold balances updated
Post conditions	Bid is added to the database and on-hold balance is updated based on auto-bids made, best bid may be updated in the item attributes
SQL	`INSERT INTO bid VALUES(item_id, curr_id, TRUE, max_bid, init_bid, "running", NOW());` or, in case of no auto-bidding: `INSERT INTO bid VALUES(item_id, curr_id, FALSE, NULL, init_bid, "running", NOW());`

12. **Update bid**

Input	Update option available only when the user is the bidder, click on update
Main success path	redirected to product details (same) page with new data displayed
Post conditions	Bid details updated
SQL	update row: `UPDATE bid SET auto_mode=new_auto, bid_limit=new_limit, bid_value=new_bid WHERE aitem_id=curr_aid AND person_id=curr_uid;`

13. **Close bidding**

Inputs	At least 1 bid must have been made so far, now click on end-sale
Preconditions	All the active bids are present in the database; at least one bid had to have been made
Main success path	If no bid available (best_bid=NULL), redirect to same page with error message, if successful: bidding ended and best-bid accepted, buyer and seller get receipts on their sale and bids, other bids are updated with rejections
Post conditions	Status of item and bids are updated accordingly in their tables
SQL	`UPDATE auction_item SET status='auctioned', close_time=NOW() WHERE aitem_id=curr_item_id;` for the buyer `SELECT * FROM bid WHERE aitem_id=curr_item_id;` then get details of each other (failed) bid so as to update bid-status send them a message `WITH rejected AS (SELECT person_id FROM bid WHERE aitem_id=curr_item_id AND person_id<>buyer_id) UPDATE bid SET status='rejected' WHERE person_id IN rejected;` and get details by `SELECT * FROM bid WHERE aitem_id=curr_item_id AND person_id<>buyer_id;`

14. **Update delivery status**

Inputs	Click on next-stage button to shift status from Auctioned⟶ Shipping⟶ Shipped and go⟶ out-for-delivery, (the next status is given from server based on previous status)
Main success path	redirect to same sale-item page, with new status displayed
Post conditions	New status is updated in the database
SQL	`UPDATE auction_item SET status=next_status WHERE aitem_id = curr_item_id;` or for direct-sale `UPDATE direct_sale_item SET status=next_status WHERE ditem_id = curr_item_id;`

15. **Buyer confirms delivery**

Inputs	buyer confirms the delivery; status should have already been at 'out-for-delivery'
Main success path	Status of the item has to be updated to delivered, transactions completed, receipts received
Post conditions	Buyer and Seller's balance has to be updated, and Buyers on-hold balance has to be updated, status update
SQL	`UPDATE auction_item SET status="delivered" WHERE aitem_id= curr_item_id;` or for direct sale use command `UPDATE direct_sale_item SET status="delivered" WHERE ditem_id = curr_item_id;`, then in the next stage get the receipt from item details `SELECT * FROM auction_item WHERE aitem_id=curr_item_id;` or `SELECT * FROM direct_sale_item WHERE ditem_id=curr_item_id;`, followed by updating buyer balance: `UPDATE person SET amount_on_hold = amount_on_hold-best_bid, balance = balance - best_bid WHERE person_id=curr_uid` , then seller balance: `UPDATE person SET balance = balance + best_bid WHERE person_id=seller_uid`

In the following screen designs are attached, with descriptions of associated use-cases for each button. If the button is repeated, the use-case is not described again.

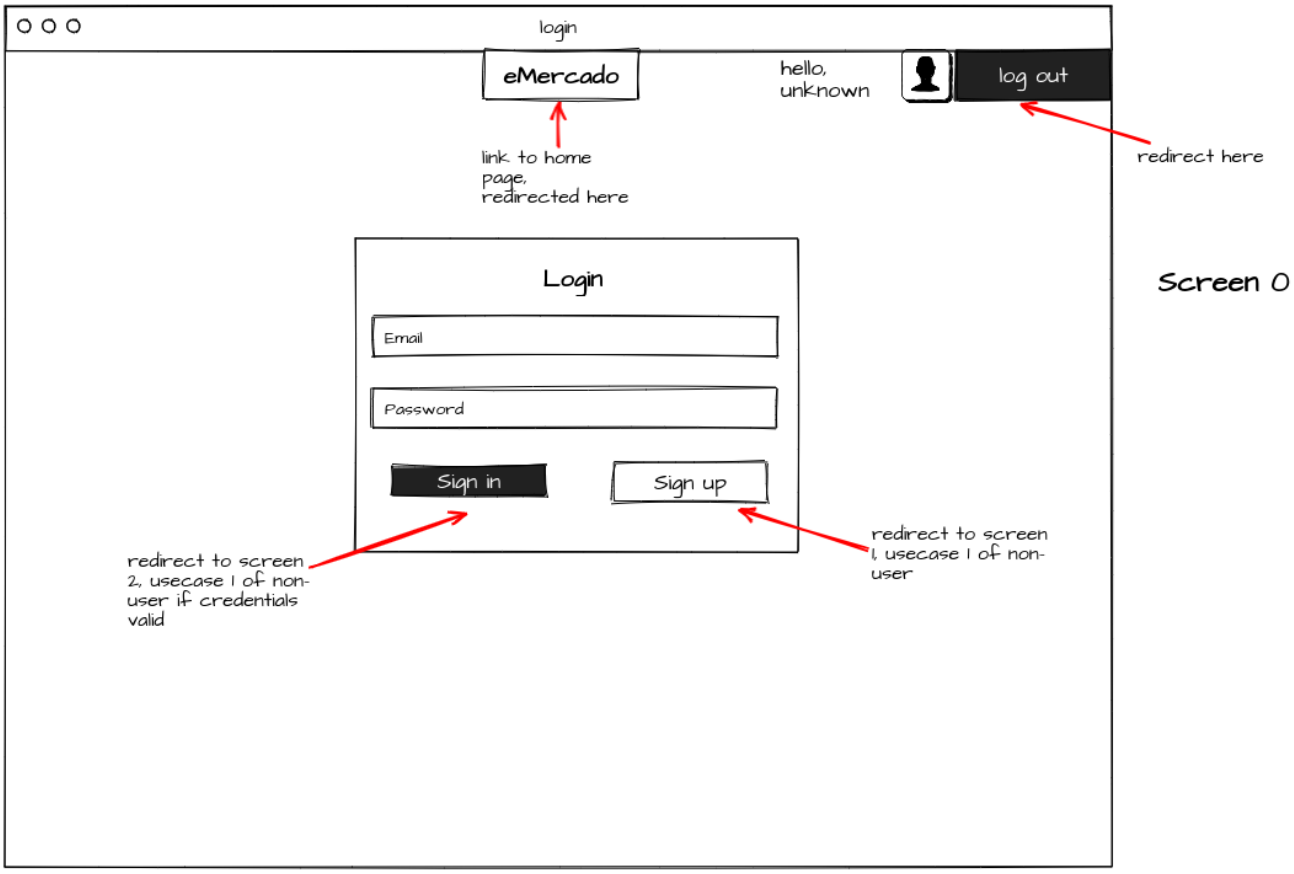

Figure 2: login screen

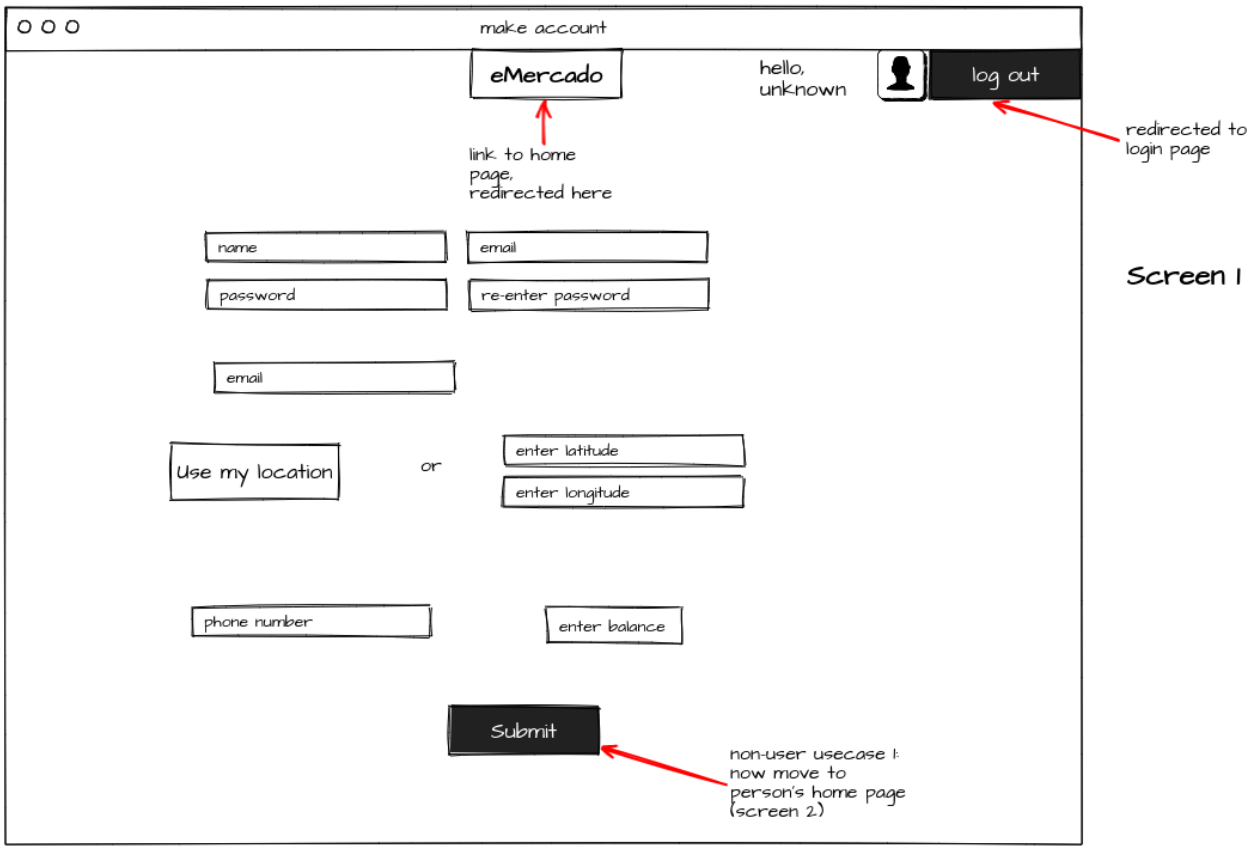

Figure 3: make account screen

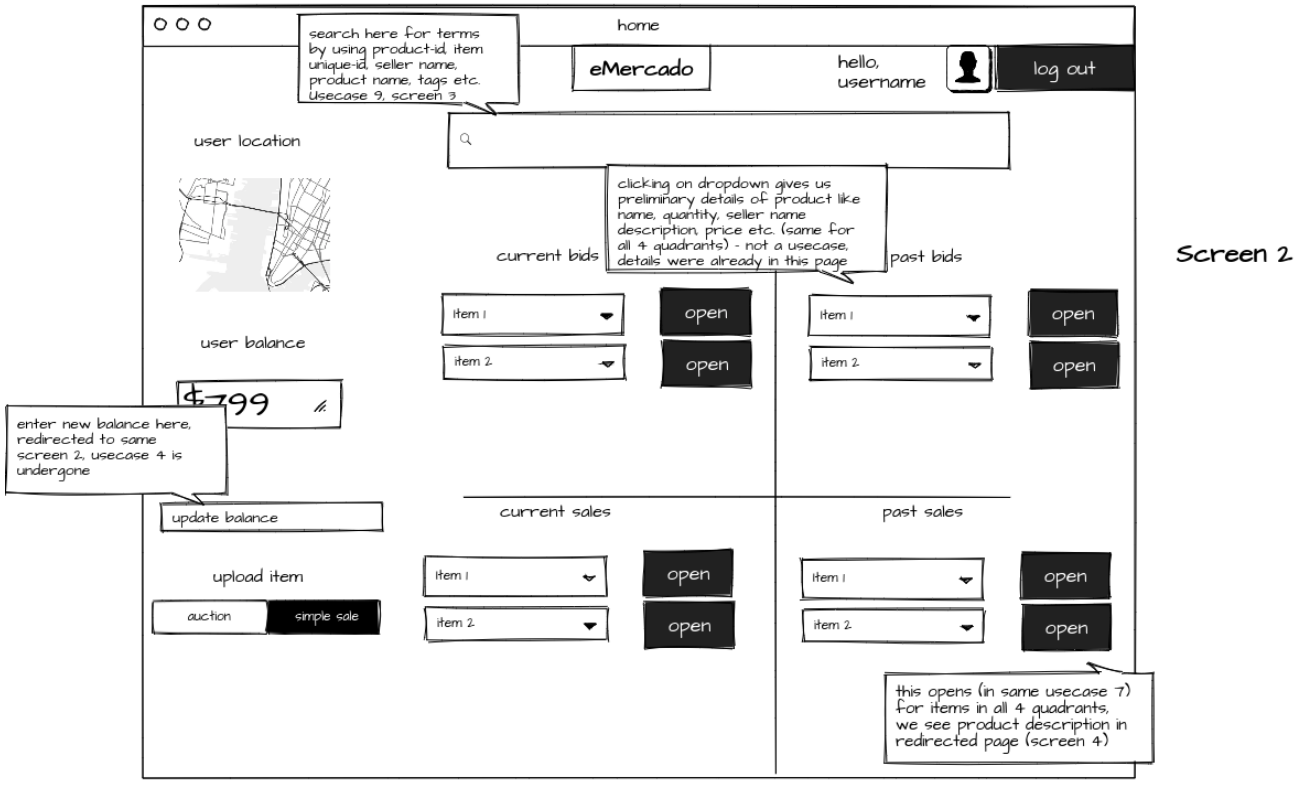

Figure 4: home screen

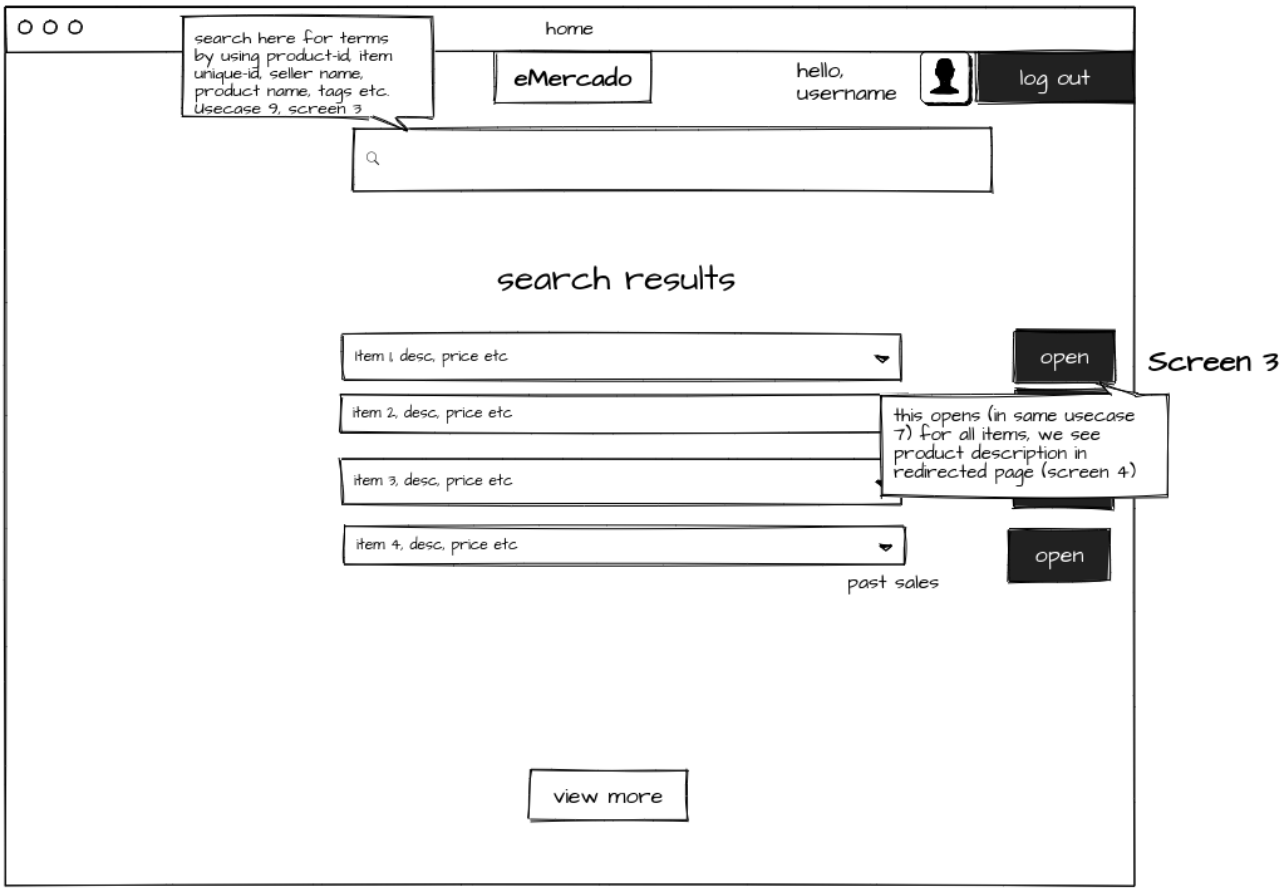

Figure 5: search screen

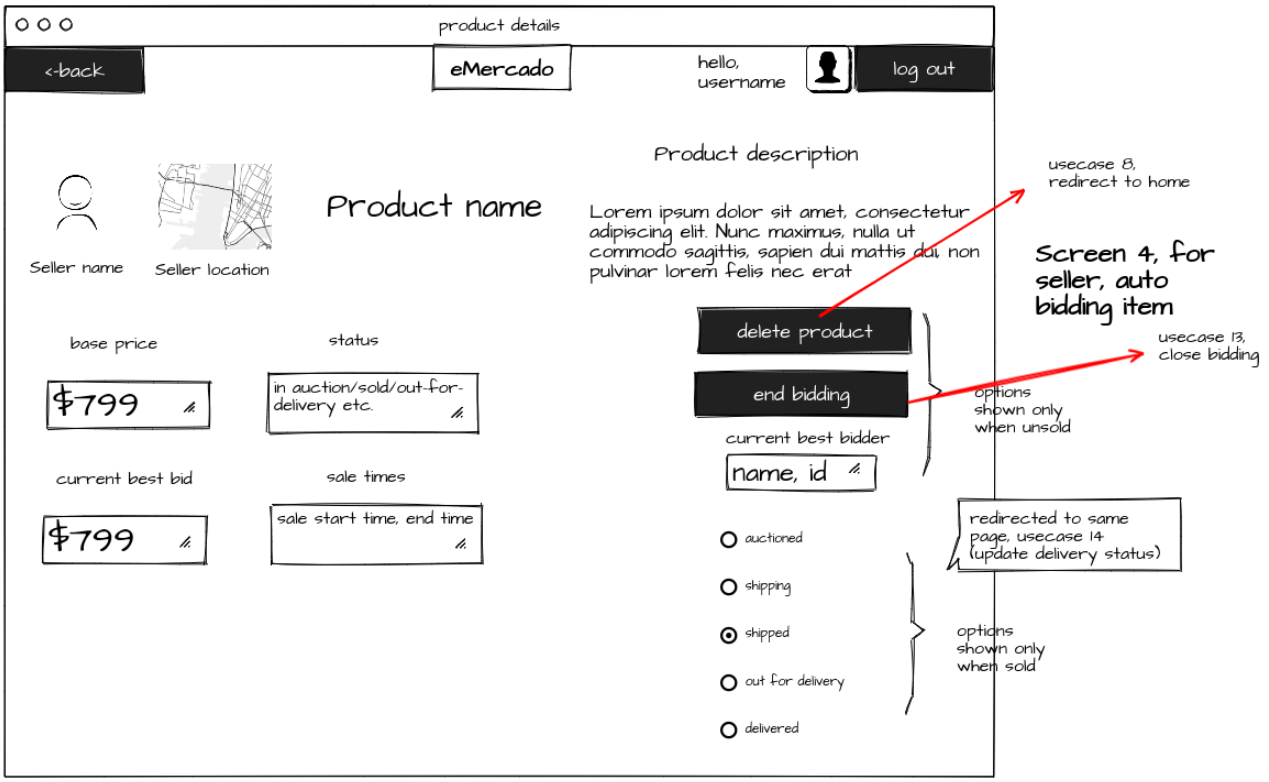

Figure 6: product-1

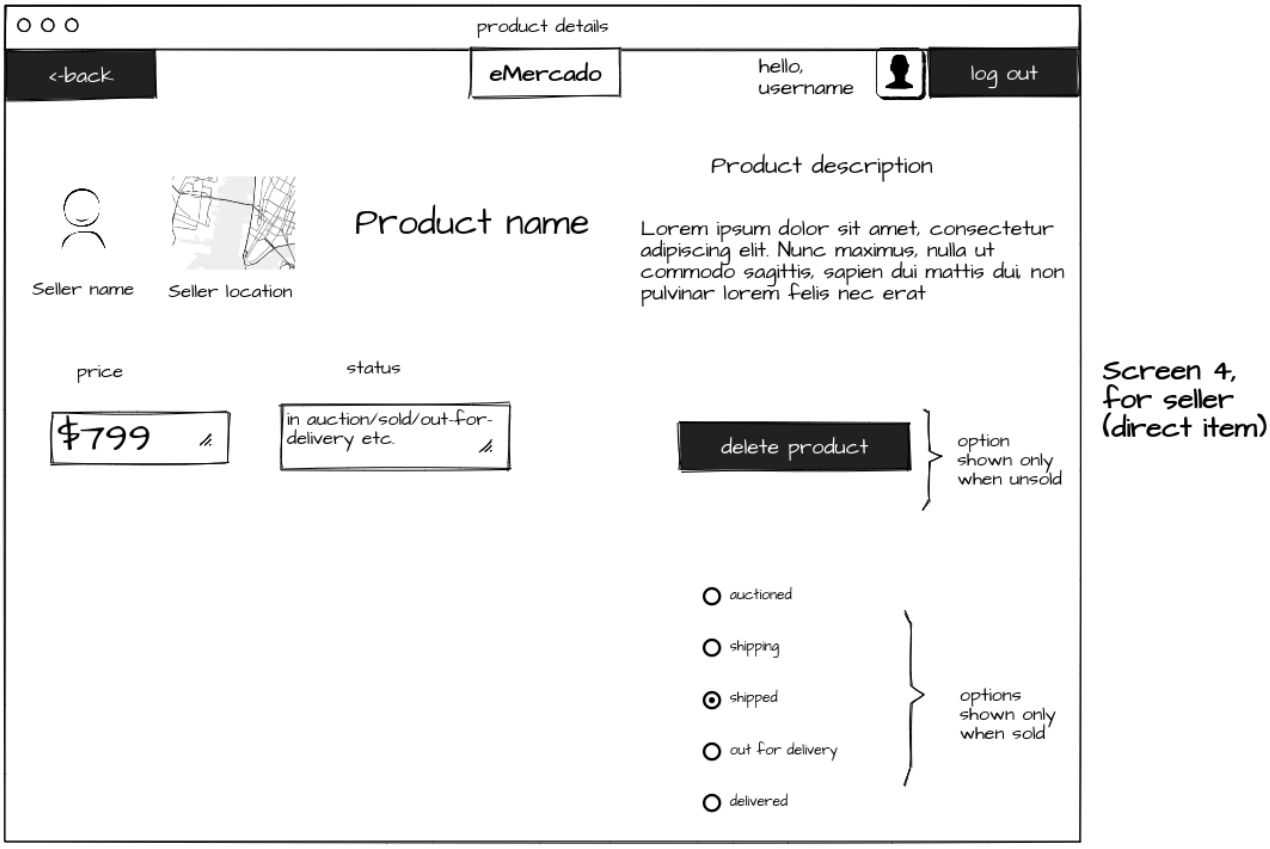

Figure 7: product-2

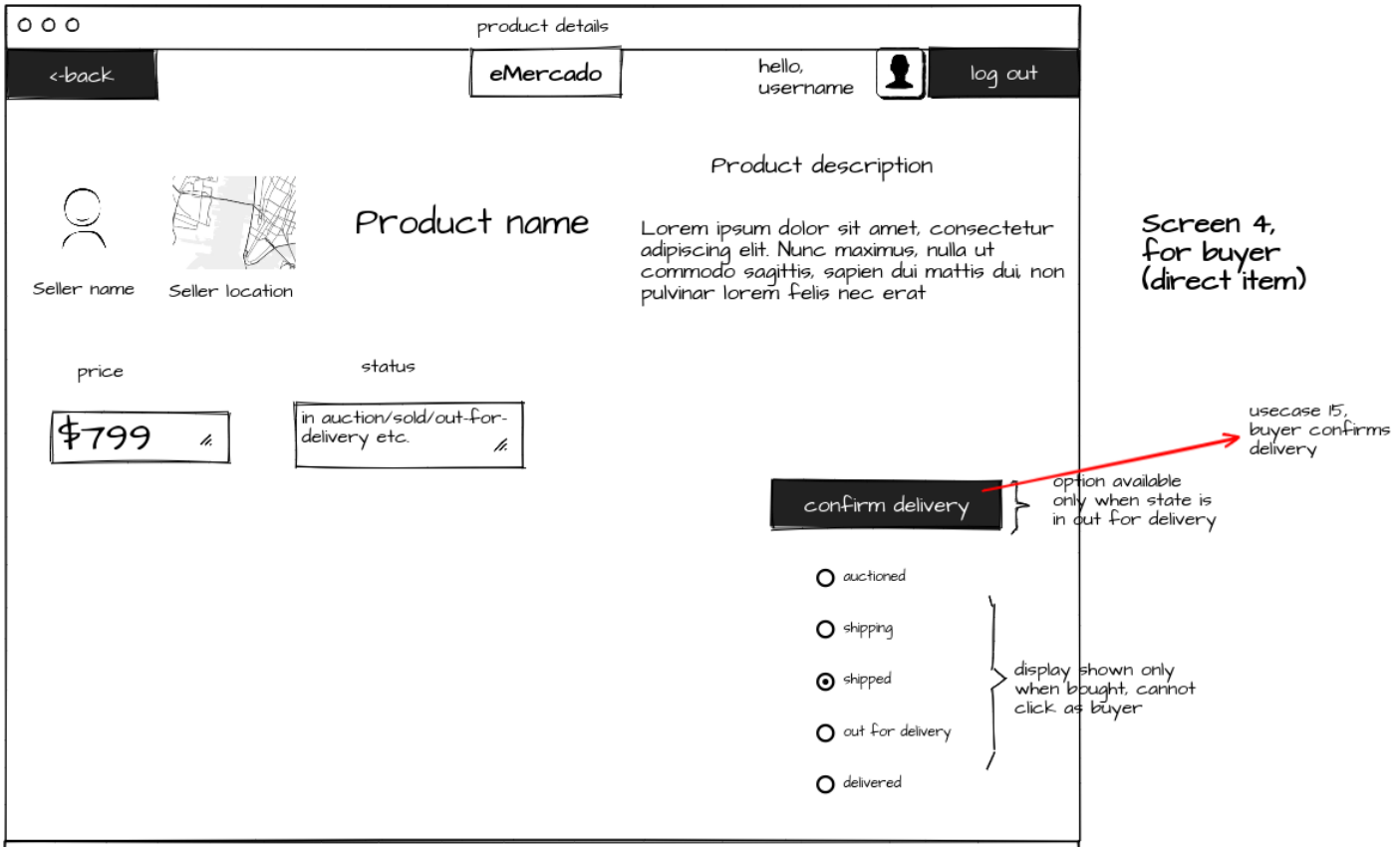

Figure 8: product-3

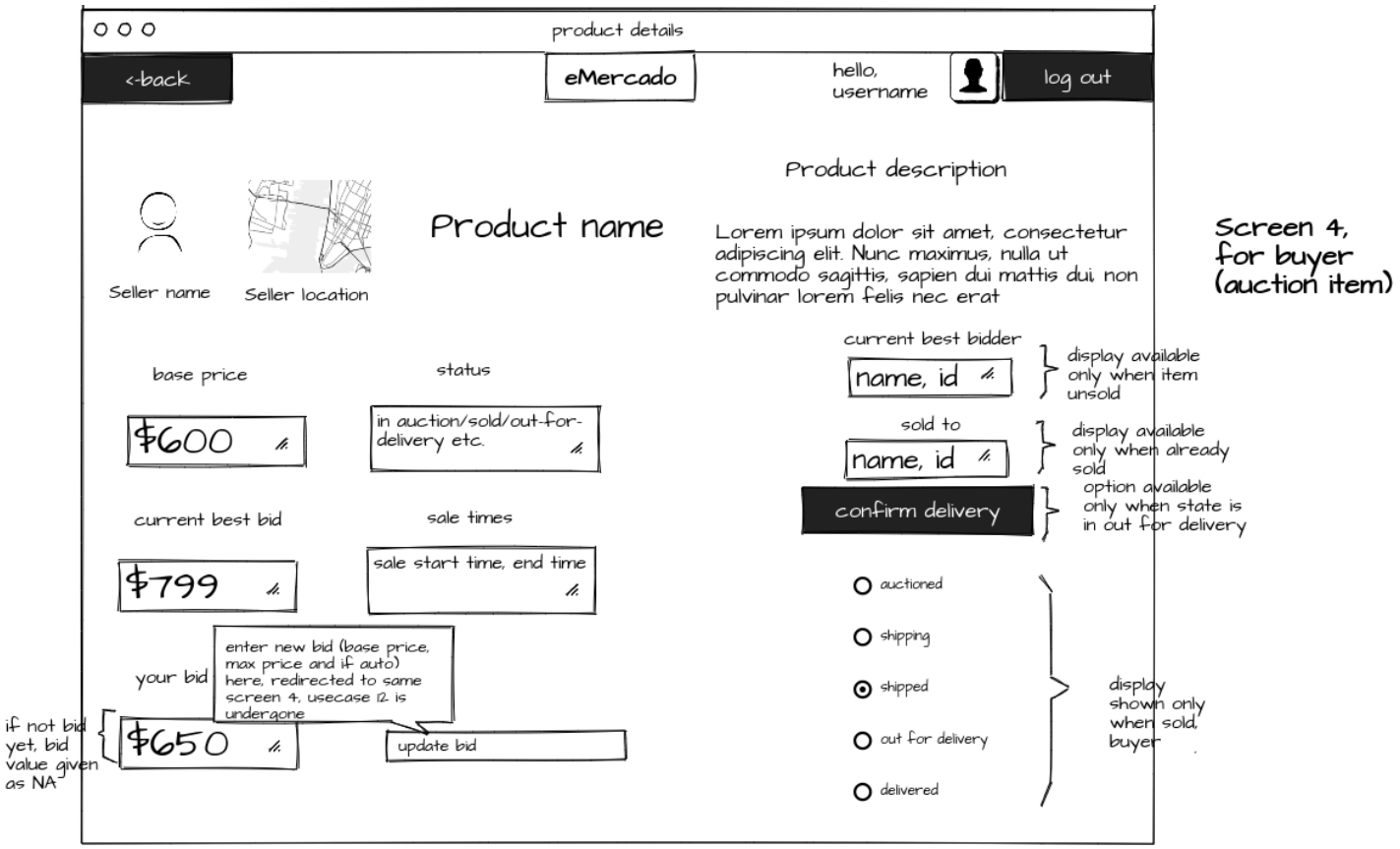

Figure 9: product-4

Appendix

Here, we have attached 3 files - eMercadoDDL.sql (which contains complete DDL for our schema), datagen.py (which is used to generate required data (csv files corresponding to relations) from downloaded data files), dataload.sql (which loads the data into the database from the csv files generated by datagen.py).

Sequence of steps to replicate the behaviour:
(1)Set the current working directory and place the 3 files - eMercadoDDL.sql, datagen.py, dataload.sql in it.
(2)Create a folder Data/ and place StateNames.csv and flipkart_com-ecommerce_sample.csv in that folder.
(3)Create new directory Datagen/ and execute the datagen.py file as follows:

```
$python3 datagen.py
```

This will create the relevant csv files in the Datagen/ folder.
(4)Enter the psql shell and move to the database. (Make sure that postgis extension is enabled).Execute the following commands to create the schema in the database and load the seed data into the database:

```
# \i eMercadoDDL.sql
```

```
# \i dataload.sql
```

And then we are done setting up a seeded version of the database.

eMercadoDDL.sql

```
DROP TABLE if exists bid;
DROP TABLE if exists direct_sale_item_tags;
DROP TABLE if exists auction_item_tags;
DROP TABLE if exists direct_sale_item;
DROP TABLE if exists auction_item;
DROP TABLE if exists person;


CREATE TABLE person (
      person_id serial,
```

```
      name varchar(25),
      email varchar(50) not null unique,
      password_hashed varchar(64),
      phone_no varchar(15),
      location geography(point,4326),
      balance float,
      amount_on_hold float check(amount_on_hold<=balance),
      primary key(person_id)
);



CREATE TABLE auction_item (
      aitem_id serial,
      identifier varchar(32),
      name varchar(150),
      description text,
      price float,
      seller_id int not null,
      status varchar(20) check(status='open' or status='closed' or
status='auctioned' or
            status='shipping' or status='shipped' or
status='out-for-delivery' or status='delivered'),
      physical_product boolean,
      quantity int default 1 check(quantity>0),
      delivery_factor float check(delivery_factor>=0),
      best_bidder int,
      best_bid float check(best_bid is null or best_bid>=price),
      start_time timestamp,
      close_time timestamp check(close_time>start_time),
      primary key(aitem_id),
      foreign key(best_bidder) references person on delete set null,
      foreign key(seller_id) references person on delete cascade
);

CREATE TABLE direct_sale_item (
      ditem_id serial,
      identifier varchar(32),
      name varchar(150),
      description text,
      price float,
      seller_id int not null,
      status varchar(25) check(status='open' or status='closed' or
```

```
status='sold' or
            status='shipping' or status='shipped' or
status='out-for-delivery' or status='delivered'),
      physical_product boolean,
      quantity int default 1 check(quantity>0),
      delivery_factor float check(delivery_factor>=0),
      buyer_id int,
      primary key(ditem_id),
      foreign key(seller_id) references person on delete cascade,
      foreign key(buyer_id) references person on delete set null
);

CREATE TABLE auction_item_tags (
      identifier int,
      tag varchar(100),
      primary key(identifier,tag),
      foreign key(identifier) references auction_item on delete cascade
);

CREATE TABLE direct_sale_item_tags (
      identifier int,
      tag varchar(100),
      primary key(identifier,tag),
      foreign key(identifier) references direct_sale_item on delete cascade
);

-- need to ensure that on-hold balances are updated when products kept ofr
sale are deleted
CREATE TABLE bid (
      aitem_id int,
      person_id int,
      auto_mode boolean,
      bid_limit float,
      bid_value float,
      status varchar(20) check(status='rejected, removed' or
status='running' or status='accepted'),
      time timestamp,
      primary key(aitem_id,person_id),
      foreign key(aitem_id) references auction_item on delete cascade,
      foreign key(person_id) references person on delete cascade
);
```

```
CREATE INDEX auslr_idx ON auction_item(seller_id);
CREATE INDEX dsslr_idx ON direct_sale_item(seller_id);
CREATE INDEX dsbyr_idx ON direct_sale_item(buyer_id);
CREATE INDEX autag_idx ON auction_item_tags(tag);
CREATE INDEX dstag_idx ON direct_sale_item_tags(tag);
```

datagen.py

```
import csv
import os
import random
from hashlib import sha256

#hashing password
h = sha256()
h.update(b'R@^d0M~@$$#0%&')

#seeding constants used
seedphnomin=6000000000
seedphnomax=9999999999
seedpassword=h.hexdigest()
seedbalance=1000.0
seedlocation='SRID=4326;POINT(0 49)'
seednumcust=10000 #updated during customer.csv construction
seedqtymax=5
seeddfmax=2
seedstarttime='2021-03-19 12:30:30'
seedendtime='2021-05-19 12:30:30'

namesizelimit = 150
tagsizelimit = 100

#filtering StateNames.csv to remove duplicate names
with open('Data/namesnoduplicates.csv', 'w') as f:
    with open('Data/StateNames.csv', mode='r') as infile:
        fields = ['person_id', 'name']
        csvwriter = csv.writer(f)
```

```
        csvwriter.writerow(fields)
        csvreader = csv.reader(infile)
        header = next(csvreader)
        seenname = set() # set for fast O(1) amortized lookup
        idx=1
        for row in csvreader:
            if(row[1] in seenname):
                continue
            csvwriter.writerow([idx,row[1]])
            seenname.add(row[1])
            idx += 1

#gen scripts

with open('Datagen/person.csv', 'w') as f:
    with open('Data/namesnoduplicates.csv', mode='r') as infile:
        fields = ['person_id', 'name', 'email', 'password_hashed',
'phone_no', 'location', 'balance', 'amount_on_hold']
        csvwriter = csv.writer(f)
        csvwriter.writerow(fields)
        csvreader = csv.reader(infile)
        header = next(csvreader)
        for row in csvreader:
            phno = random.randint(seedphnomin,seedphnomax)

csvwriter.writerow([row[0],row[1],row[1]+'@emercado.com',seedpassword
,phno,seedlocation,seedbalance,0.0])
            if(seednumcust<=int(row[0])):
                break

with open('Datagen/auction_item.csv', 'w') as f:
    with open('Data/flipkart_com-ecommerce_sample.csv', mode='r') as
infile:
        fields =
['aitem_id','identifier','name','description','price','seller_id','st
atus','physical_product','quantity','delivery_factor','best_bidder','
best_bid','start_time','close_time']
        csvwriter = csv.writer(f)
        csvwriter.writerow(fields)
```

```
        csvreader = csv.reader(infile)
        header = next(csvreader)
        idx = 1
        for row in csvreader: #discounted price is used as (base)
price
            if (row[7]==""):
                continue
            if (len(row[3])>namesizelimit):
                continue
            custid = random.randint(1,seednumcust)
            qty = random.randint(1,seedqtymax)
            df = random.randint(0,seeddfmax)

csvwriter.writerow([idx,row[0],row[3],row[10].replace('\n', '
').replace('"','').replace('\'',''),row[7],custid,'open','true',qty,d
f,'NULL','NULL',seedstarttime,seedendtime])
            idx += 1

with open('Datagen/direct_sale_item.csv', 'w') as f:
    with open('Data/flipkart_com-ecommerce_sample.csv', mode='r') as
infile:
        fields =
['aitem_id','identifier','name','description','price','seller_id','st
atus','physical_product','quantity','delivery_factor','buyer_id']
        csvwriter = csv.writer(f)
        csvwriter.writerow(fields)
        csvreader = csv.reader(infile)
        header = next(csvreader)
        idx = 1
        for row in csvreader: #retail price is used as price
            if (row[6]==""):
                continue
            if (len(row[3])>namesizelimit):
                continue
            custid = random.randint(1,seednumcust)
            qty = random.randint(1,seedqtymax)
            df = random.randint(0,seeddfmax)

csvwriter.writerow([idx,row[0],row[3],row[10].replace('\n', '
```

```
').replace('"','').replace('\'',''),row[6],custid,'open','true',qty,d
f,'NULL'])
            idx += 1


with open('Datagen/auction_item_tags.csv', 'w') as f:
    with open('Data/flipkart_com-ecommerce_sample.csv', mode='r') as
infile:
        fields = ['aitem_id','tag']
        csvwriter = csv.writer(f)
        csvwriter.writerow(fields)
        csvreader = csv.reader(infile)
        header = next(csvreader)
        idx = 1
        for row in csvreader:
            if (row[6]==""):
                continue
            if (len(row[3])>namesizelimit):
                continue
            cattree = row[4]
            catlist = cattree.strip('["] ').split(" >> ")
            seenpair = set()
            for cat in catlist:
                if cat in seenpair:
                    continue
                if (len(cat)>tagsizelimit):
                    continue
                csvwriter.writerow([idx,cat])
                seenpair.add(cat)
            idx += 1

with open('Datagen/direct_sale_item_tags.csv', 'w') as f:
    with open('Data/flipkart_com-ecommerce_sample.csv', mode='r') as
infile:
        fields = ['aitem_id','tag']
        csvwriter = csv.writer(f)
        csvwriter.writerow(fields)
        csvreader = csv.reader(infile)
        header = next(csvreader)
```

```
        idx = 1
        for row in csvreader:
            if (row[6]==""):
                continue
            if (len(row[3])>namesizelimit):
                continue
            cattree = row[4]
            catlist = cattree.strip('["] ').split(" >> ")
            seenpair = set()
            for cat in catlist:
                if cat in seenpair:
                    continue
                if (len(cat)>tagsizelimit):
                    continue
                csvwriter.writerow([idx,cat])
                seenpair.add(cat)
            idx += 1
```

dataload.sql

```
 \copy public.person FROM './Datagen/person.csv' DELIMITER ',' CSV HEADER
NULL 'NULL' QUOTE '"' ESCAPE '''';
 \copy public.auction_item FROM './Datagen/auction_item.csv' DELIMITER ','
CSV HEADER NULL 'NULL' QUOTE '"' ESCAPE '''';
 \copy public.direct_sale_item FROM './Datagen/direct_sale_item.csv'
DELIMITER ',' CSV HEADER NULL 'NULL' QUOTE '"' ESCAPE '''';
 \copy public.auction_item_tags FROM './Datagen/auction_item_tags.csv'
DELIMITER ',' CSV HEADER NULL 'NULL' QUOTE '"' ESCAPE '''';
 \copy public.direct_sale_item_tags FROM
'./Datagen/direct_sale_item_tags.csv' DELIMITER ',' CSV HEADER NULL 'NULL'
QUOTE '"' ESCAPE '''';
```
