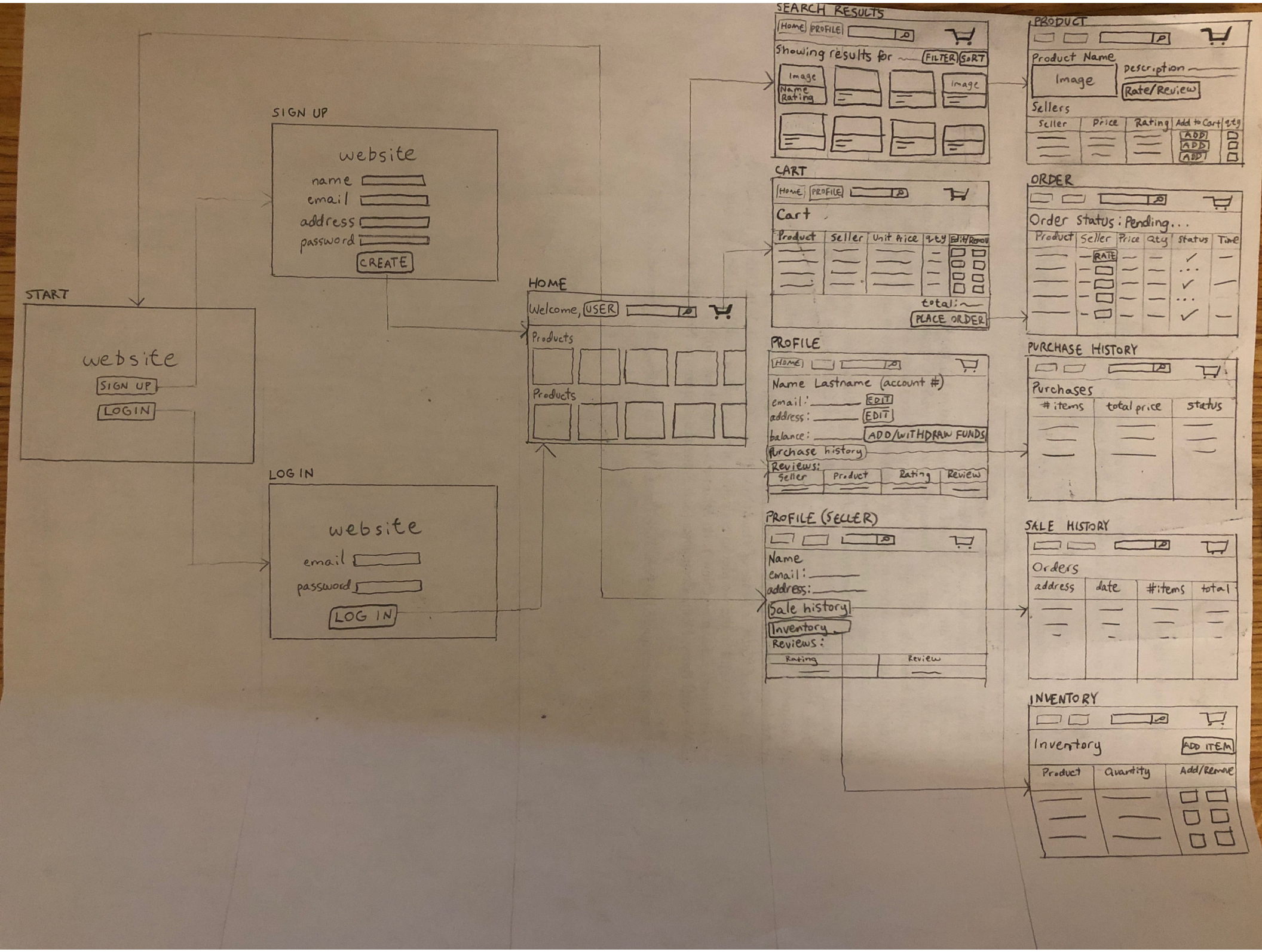
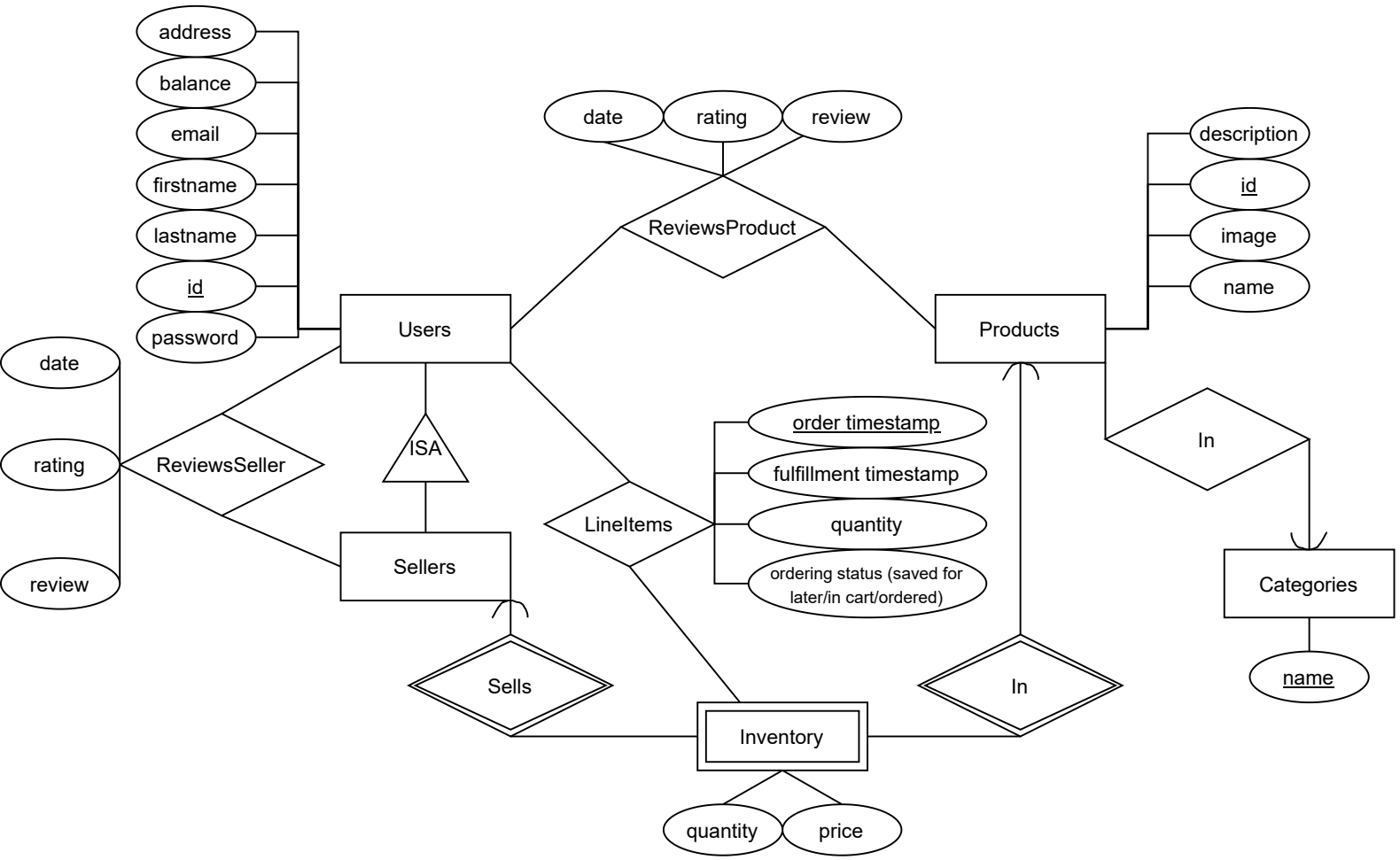

START
website
SIGN UP
LOGIN
SIGN UP
website
name
email
address
password
CREATE
LOG IN
website
email
password
LOG IN
HOME
Welcome, USER
Products
Products
SEARCH RESULTS
HOME
PROFILE
Showing results for
FILTER
SORT
Image
Name
Rating
Image
CART
HOME
PROFILE
Cart
Product
Seller
Unit Price
qty
Edit/Remove
total:
PLACE ORDER
PROFILE
HOME
Name Lastname (account #)
email:
EDIT
address:
EDIT
balance:
ADD/WITHDRAW FUNDS
Purchase history
Reviews:
Seller
Product
Rating
Review
PROFILE (SELLER)
Name
email:
address:
Sale history
Inventory
Reviews:
Rating
Review
PRODUCT
Product Name
Image
description
Rate/Review
Sellers
Seller
Price
Rating
Add to Cart
qty
ADD
ADD
ADD
ORDER
Order status: Pending...
Product
Seller
Price
Qty
Status
Time
RATE
PURCHASE HISTORY
Purchases
items
total price
status
SALE HISTORY
Orders
address
date
#items
total
INVENTORY
Inventory
ADD ITEM
Product
Quantity
Add/Remove


address
balance
email
firstname
lastname
id
password
Users
date
rating
review
ReviewsProduct
description
id
image
name
Products
date
rating
review
ReviewsSeller
ISA
Sellers
order timestamp
fulfillment timestamp
quantity
ordering status (saved for later/in cart/ordered)
LineItems
In
Categories
name
Sells
In
Inventory
quantity
price

E/R Diagram: Tables, Constraints, Assumptions

Assumptions

1. One user cannot place two orders simultaneously.
2. Furthermore, assume that no two users can place orders simultaneously. That is, we can determine a time-based ordering of every order such that we can decrement and check inventories and balances before the next order takes place.
3. Assume that two users can live at the same address.
4. A specific product cannot be saved for later and in the cart at the same time.
5. Assume each user is associated with only one address.
6. Assume that for any review, a user must leave a rating but is not required to leave a write-up

Tables:

Users(id, balance, email, firstname, lastname, password, address)

Sellers(id)

- Note: We use the "E/R Style" of subclass translation here.

Products(id, image, description, name, category)

- Note: Price is not included as an attribute here, as it will be included in the inventory table. We do this to allow sellers to offer the same (or marginally different) products at different price points.
- Here, we merge the Products table with the ProductCategories relationship. We do this to simplify our table declarations, because every product must be in one category.

Categories(name)

- Table of predefined categories that products must reference

ReviewsSeller(User_id, Seller_id, date, rating, review)

- Here, we have a many-many relationship, as buyers can review multiple sellers, and sellers can be reviewed by multiple buyers. This relationship set enforces the constraint that each user can review a specific seller only once, as the key is: {User_id, Seller_id}. Therefore, for a given (user, seller) pair, there is only one tuple in the table.

ReviewsProduct(User_id, Product_id, date, rating, review)

- Similar to the *ReviewsSeller* table, we have a many-many relationship here. Again, the relationship enforces the constraint that each user can review a specific product only once.

Inventory(Seller_id, Product_id, quantity, price)

- Keeps track of the price and quantity of each specific product that a given seller owns.

LineItems(User_id, Seller_id, Product_id, order timestamp, fulfilment timestamp, quantity, ordering status)

- **Definition of less obvious attributes:**

    - Order Timestamp: A timestamp that keeps track of when a user places an order for a given product. If an order has not been placed yet, this attribute may be NULL. All LineItems in the same order will have the same timestamp, enabling aggregate functions.
    - Fulfillment Timestamp: Keeps track of when an order has been fulfilled. If NULL, the order has not been fulfilled.
    - Ordering Status: Keeps track of whether a line item has been ordered, saved for later, or is currently in the cart.
- This table keeps a record of all line items that are either currently in a user's cart, have been saved for later, or have already been ordered. It is a many-many relationship, so it takes part of its key from the *User* table and part of its key from the *Inventory* table. It takes the last part of its key from the order timestamp. From the assumptions section, a user cannot place two orders simultaneously and a product cannot be concurrently saved for later and in a user's cart, thus these 4 attributes are a unique tuple identifier.

Constraints:

1. Not NULL Constraints:

a) *Users:* [address, balance, email, id, firstname, lastname, password]
    i. *Sellers:* [id] – this will be enforced by a FOREIGN KEY to *Users*
b) *Products:* [description, id, image, name]
c) *Categories*: [name]
d) *ReviewsSeller:* [date, rating]
    i. User_id and Seller_id are already enforced as Not NULL, as they will be FOREIGN KEYs
    ii. Note that this means a user could leave a rating with no review. A user cannot leave a review with no rating.
e) *ReviewsProduct:* [date, rating]
    i. User_id and Product_id are already enforced as Not NULL, as they will be FOREIGN KEYs
    ii. Note that this means a user could leave a rating with no review. A user cannot leave a review with no rating.
f) *Inventory:* [quantity, price]
    i. If a seller has a product for sale, they must have some quantity of it and know what price they are selling it for. Note that the quantity can be 0, but it cannot be negative or NULL.
g) *LineItems:* [quantity, ordering status]
    i. In order to appear as a line item, the User must have some previous interaction with a product (i.e. put it in the cart, save it for later, or purchase it). This idea requires us to always know the quantity and order status of a LineItem.
    ii. The order and fulfillment timestamps can be NULL, when an order hasn't been placed or an order hasn't been fulfilled respectively.

2. **Primary Keys:**

    a) *Users*: {id}
    b) *Sellers*: {id}
    c) *Products:* {id}
    d) *Categories:* {name}
    e) *ReviewsSeller:* {User_id, Seller_id}
    f) *ReviewsProduct:* {User_id, Product_id}
    g) *Inventory:* {Seller_id, Product_id}
    h) *LineItems*: {User_id, Seller_id, Product_id, order timestamp}

3. **Unique Keys**

    a) *User*: {email}
    b) All other tables: N/A

4. **Foreign Keys**

    a) *Users:* N/A
    b) *Sellers*: {User_id}
    c) *Products*: {Category_name}
    d) *Categories*: N/A
    e) *ReviewsSeller:* {User_id, Seller_id}
    f) *ReviewsProduct:* {User_id, Product_id}
    g) *Inventory:* {Seller_id, Product_id}
    h) *LineItems:* {User_id, Seller_id, Product_id}

5. **Additional Checks:**

    a) Create a trigger before INSERT on User:
        i. If New.User_Id not in User, and New.Balance <> 0, SET New.Balance = 0
        ii. Here, we enforce that a User's initial balance is 0
    b) If an order reduces a user's balance below 0, reject the order
    c) If an order reduces a seller's inventory of any products below 0, reject the order
    d) Introduce a way of restricting Users from only reviewing products that they have purchase
    e) Create two triggers before INSERT/UPDATE on *Reviews* tables:
        i. IF New.rating > 5, SET New.rating = 5
        ii. IF New.rating < 0, SET New.rating = 0
    f) Create a view *SellerOrders* on (LineItems NATURAL JOIN Users), Seller:
        i. SELECT all LineItems that have been ordered, and group by seller
        ii. This will allow sellers to check the orders that have been fulfilled or need to be fulfilled (i.e. when fulfillment status IS NULL)
        iii. This also allows buyers to sort through their orders
    g) Create a trigger on LineItems before UPDATE
        i. If the old order timestamp IS NOT NULL, then RAISE EXCEPTION

ii. This stops buyers and sellers from updating orders once they have been made
iii. We still need a way to stop prices from changing once an order has been placed, as LineItems does not contain pricing information