Project 1 - Auction Simulation

2020310083 Hyungjun Shon

Dept. of System Management Engineering
Dept. of Software
Sungkyunkwan University

November 13, 2024

Contents

1 Introduction

This Auction System application is designed to facilitate online bidding, selling, and purchasing of items. The application allows users to list items for auction, place bids, and manage accounts. Sellers can categorize items, set bid closing dates, and manage sales, while buyers can bid on items, check their bidding status, and finalize purchases. Administrators have additional access to oversee user accounts, view transaction summaries, and review marketplace statistics.

The primary components include user management, item listings, bidding, billing, and administrative functions, with PostgreSQL used as the backend database. Each functionality is linked to a specific SQL query, which ensures the seamless operation of the auction process.

2 ER model

2.1 ER Diagram

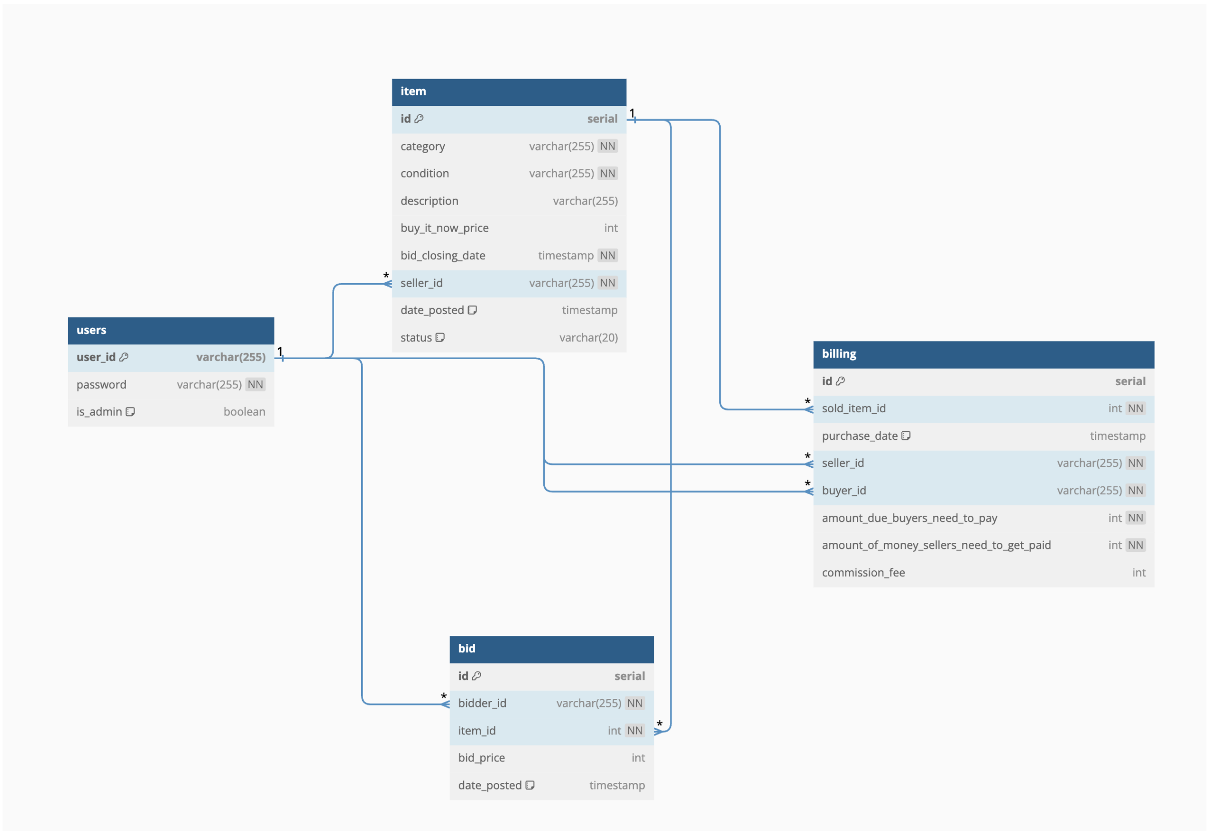

Figure 1: ER Diagram

2.2 Drop all existing tables

```sql
DROP TABLE IF EXISTS billing;
DROP TABLE IF EXISTS bid;
DROP TABLE IF EXISTS item;
DROP TABLE IF EXISTS users;
```

Drop all existing tables to ensure a clean database schema.

2.3 Users Table

```sql
create table users (
    user_id varchar(255) primary key,
    password varchar(255) not null,
    is_admin boolean default false
);
```

Purpose:

Stores user information, with roles distinguishing between admin and regular users.

Attributes:

- `user_id`: Unique identifier for each user, stored as a varchar (primary key).
- `password`: User's password, stored as a varchar.
- `is_admin`: Boolean flag indicating if the user is an administrator (`false` by default).

2.4 Item Table

```sql
create table item (
    id serial primary key,
    category varchar(255) not null,
    condition varchar(255) not null,
    description varchar(255),
    buy_it_now_price int check (buy_it_now_price > 0),
    bid_closing_date timestamp not null,
    seller_id varchar(255) not null,
    date_posted timestamp default current_timestamp,
    status varchar(20) DEFAULT 'AVAILABLE' CHECK (status IN ('AVAILABLE', 'SOLD', 'EXPIRED')),

    constraint fk_seller_id foreign key(seller_id) references users(user_id) on delete cascade
);
```

Purpose:

Manages items listed for sale, with details on category, condition, and status.

Attributes:

- `id`: Unique identifier for each item (serial, primary key).
- `category`: Type of item, constrained to predefined categories (e.g., `ELECTRONICS`, `BOOKS`) and not null.
- `condition`: Condition of the item, with predefined values (`NEW`, `LIKE_NEW`, etc.) and not null.
- `description`: Text description of the item.
- `buy_it_now_price`: Price for immediate purchase, must be positive.
- `bid_closing_date`: Timestamp for when bidding closes on the item and is not null.
- `seller_id`: Foreign key linking to the seller in `users` table.
- `date_posted`: Timestamp for when the item was posted, defaults to current time.
- `status`: Current status of the item (`AVAILABLE`, `SOLD`, `EXPIRED`), defaulting to `AVAILABLE`.

2.5 Bid Table

```sql
create table bid (
    id serial primary key,
    bidder_id varchar(255) not null,
    item_id int not null,
    bid_price int check (bid_price > 0),
    date_posted timestamp default current_timestamp,

    constraint fk_item_id foreign key(item_id) references item(id) on delete cascade,
    constraint fk_bidder_id foreign key(bidder_id) references users(user_id) on delete cascade
);
```

Purpose:

Tracks each bid made on items.

Attributes:

- id: Unique identifier for each bid (serial, primary key).
- bidder_id: Foreign key linking to the bidder in the users table. (cascade option for delete)
- item_id: Foreign key linking to the item being bid on. (cascade option for delete)
- bid_price: Amount bid, must be positive.
- date_posted: Timestamp for when the bid was placed, defaults to current time.

2.6 Billing Table

```sql
create table billing (
    id serial primary key,
    sold_item_id int not null unique,
    purchase_date timestamp default current_timestamp,
    seller_id varchar(255) not null,
    buyer_id varchar(255) not null,
    amount_due_buyers_need_to_pay int not null check (amount_due_buyers_need_to_pay >= 0),
    amount_of_money_sellers_need_to_get_paid int not null check
 ↪  (amount_of_money_sellers_need_to_get_paid >= 0),
    commission_fee int,

    constraint fk_sold_item_id foreign key(sold_item_id) references item(id) on delete cascade,
    constraint fk_seller_id foreign key(seller_id) references users(user_id) on delete cascade,
    constraint fk_buyer_id foreign key(buyer_id) references users(user_id) on delete cascade
);

create or replace function calculate_commission_fee()
returns trigger as $$
begin
    new.commission_fee := floor(new.amount_due_buyers_need_to_pay * 0.1)::int

    if new.commission_fee < 1 then
        new.commission_fee := 0;
    end if;

    new.amount_of_money_sellers_need_to_get_paid := new.amount_due_buyers_need_to_pay -
     ↪  new.commission_fee;
    return new;
end;
$$ language plpgsql;

create trigger trg_calculate_commission_fee
before insert or update on billing
for each row
execute function calculate_commission_fee();
```

Purpose:

Handles payment details and commission for sold items.

Attributes:

- id: Unique identifier for each billing entry (serial, primary key).
- sold_item_id: Foreign key linking to the sold item in the item table. (cascade option for delete)
- purchase_date: Timestamp for when the item was purchased, defaults to current time.
- seller_id: Foreign key linking to the seller in users table. (cascade option for delete)
- buyer_id: Foreign key linking to the buyer in users table. (cascade option for delete)
- amount_due_buyers_need_to_pay: Total amount buyer needs to pay, must be non-negative.
- amount_of_money_sellers_need_to_get_paid: Amount the seller receives after commission, must be non-negative.

- commission_fee: Calculated commission fee based on 10% of purchase amount, rounded down.

Trigger and Function:

- calculate_commission_fee: A function that calculates the 10% commission fee, rounded down.
- trg_calculate_commission_fee: Trigger to automatically apply the commission fee calculation on insert or update of billing entries.

3 Compile and Run

3.1 Prerequisites

- postgresql-42.6.0.jar: JDBC driver for PostgreSQL.
- ddl.sql: SQL script to create the database schema.
- data.sql: SQL script to insert sample data into the tables.
- Auction.Java: Main application file.
- Makefile: Script to compile and run the application.

3.2 create schema and insert sample data

```
psql -U s20310083 # login to the database
\c s20310083 # connect to the database s20310083
\i ddl.sql # create schema
\i data.sql # insert sample data
```

3.3 Compile and Run

```
make all # compile the application
make run # run the application
```

4 Functionality and SQL Queries explanation

NOTE All terminal outputs are formatted in terminal (report does not endure the width)

4.1 User Management

4.1.1 Login as normal user

```
SELECT
    *
FROM
    users
WHERE
    user_id = ?
    AND password = ?;
```

Purpose:

Authenticate a user based on the provided user_id and password.

Parameters:

- user_id: User's login id.
- password: User's password.

- Return User information if the credentials match, otherwise an empty result set.

handling:

- If the query returns a result, the user is successfully authenticated.
- If the query returns an empty result, the login attempt is rejected.

4.1.2 Sign up as a new user

```sql
INSERT INTO users
    (user_id, password, is_admin)
VALUES
    (?, ?, ?);
```

Purpose:

Register a new user with the provided user_id, password, and admin status.

Parameters:

- `user_id`: User's login id.
- `password`: User's password.
- `is_admin`: Boolean flag indicating if the user is an administrator.

Handling:

- If the query executes successfully, the user is registered.
- If the user_id is already taken, the pre selection query check if the user_id is already taken and fail.

4.1.3 Login as an administrator

```sql
SELECT
    is_admin
FROM
    users
WHERE
    user_id = ?
    AND password = ?;
```

Purpose:

Authenticate an administrator based on the provided user_id and password.

Parameters:

- `user_id`: Admin's login id.
- `password`: Admin's password.
- Return Admin status if the credentials match, otherwise an empty result set.

Handling:

- If the query returns a result, the admin is successfully authenticated.
- If the query returns an empty result, the login attempt is rejected.

4.2 Selling Items

```
INSERT INTO item
    (category, condition, description, buy_it_now_price, bid_closing_date, seller_id)
VALUES
    (?, ?, ?, ?, ?, ?);
```

Purpose:

Create a new item listing with the provided details.

Parameters:

- `category`: Type of item being listed.
- `condition`: Condition of the item.
- `description`: Text description of the item. (item name or details)
- `buy_it_now_price`: Price for immediate purchase.
- `bid_closing_date`: Timestamp for when bidding closes.
- `seller_id`: ID of the user listing the item. (automatically set by the current user)

Handling:

- If the query executes successfully, the item is listed for sale.
- If any of the required fields are missing or invalid, the addition will fail.

4.3 View the list of items that current user is selling

4.3.1 View the list of items that current user is selling

```
SELECT i.id AS item_id,
       i.description,
       i.condition,
       b.bidder_id AS buyer_id,
       b.bid_price AS bidding_price,
       b.date_posted AS bidding_date
FROM item i
LEFT JOIN bid b ON i.id = b.item_id
WHERE i.seller_id = ?
ORDER BY i.id, b.bid_price DESC
```

Purpose:

Retrieve all items listed by the current user and the corresponding bids. (sold item is included too)

Parameters:

- `seller_id`: ID of the user selling the items (automatically set by the current user).
- Return List of items with their details and bids.

Handling

- If the query returns a result, the items are displayed to the user.
- If the query returns an empty result, the user has no items listed for sale.

4.4 Buy Items

4.4.1 List all available items for bidding with certain filters

```sql
SELECT i.*,
       EXTRACT(EPOCH FROM (i.bid_closing_date - CURRENT_TIMESTAMP)) / 60 AS minutes_left,
       b.highest_bid,
       u.user_id AS highest_bidder
FROM item i
LEFT JOIN (
    SELECT item_id,
           MAX(bid_price) AS highest_bid,
           (SELECT bidder_id
            FROM bid
            WHERE bid_price = (SELECT MAX(bid_price) FROM bid WHERE item_id = b.item_id)
            AND item_id = b.item_id
            LIMIT 1) AS highest_bidder_id
    FROM bid b
    GROUP BY item_id
) b ON i.id = b.item_id
LEFT JOIN users u ON b.highest_bidder_id = u.user_id
WHERE i.status = 'AVAILABLE'
  AND i.bid_closing_date > CURRENT_TIMESTAMP
  -- Optional filters
  [AND i.category = ?]              -- Only if category is provided
  [AND i.condition = ?]             -- Only if condition is provided
  [AND i.description ILIKE ?]       -- Only if keyword is provided, with keyword formatted as
↪  '%keyword%'
  [AND i.seller_id = ?]             -- Only if seller is specified and is not 'any'
  [AND i.date_posted >= ?]          -- Only if datePosted is provided

GROUP BY i.id,
         b.highest_bid,
         u.user_id;
```

Purpose:

Retrieve available items based on various search criteria.

Parameters:

- category: Type of item to filter by (optional).
- condition: Condition of the item to filter by (optional).
- description: Keyword to search for in the item description (optional).
- seller_id: ID of the seller to filter by (optional).
- date_posted: Minimum date posted to filter by (optional).
- Return List of items matching the search criteria.

Handling:

- If the query returns a result, the items are displayed to the user.
- If the query returns an empty result, there are no items matching the search criteria.

4.4.2 Place a bid on an item

```sql
-- Retrieve the current highest bid for the specified item
SELECT COALESCE(MAX(bid_price), 0) AS highest_bid
FROM bid
WHERE item_id = ?


```

```sql

-- Get the "Buy It Now" price and seller ID for the specified item
SELECT buy_it_now_price, seller_id
FROM item
WHERE id = ?



-- Insert a bid with the bid price (if bid exceeds "Buy It Now" price, set bid price to "Buy It
↪  Now" price)
INSERT INTO bid (bidder_id, item_id, bid_price)
VALUES (?, ?, ?)


-- if bid as buy it now price, update status to SOLD and insert billing information
INSERT INTO billing (sold_item_id, seller_id, buyer_id, amount_due_buyers_need_to_pay)
VALUES (?, ?, ?, ?)


UPDATE item
SET status = 'SOLD'
WHERE id = ?
```

Purpose:

Place a bid on an item and handle the bidding process.

Parameters:

- `item_id`: ID of the item being bid on.
- `bid_price`: Amount of the bid.

Handling:

- If bid price is less than the current highest bid, the bid is rejected.
- If the bid is successful, the bid is recorded.
- If the bid exceeds the “Buy It Now” price, the item is marked as sold and billing information is added.

4.5 Check the status of current user’s bids

```sql
-- temporary table
WITH UserBids AS (
    SELECT DISTINCT ON (item_id) item_id,
           bid_price AS your_bid
    FROM bid
    WHERE bidder_id = ?
    ORDER BY item_id, date_posted DESC
),
MaxBids AS (
    SELECT b.item_id,
           b.bid_price AS highest_bid,
           b.bidder_id AS highest_bidder
    FROM bid b
    INNER JOIN (
        SELECT item_id,
               MAX(bid_price) AS max_bid_price
        FROM bid
        GROUP BY item_id
    ) max_bids
    ON b.item_id = max_bids.item_id
   AND b.bid_price = max_bids.max_bid_price
    ORDER BY b.item_id, b.date_posted ASC
)
SELECT i.id AS item_id,
       i.description,
       i.bid_closing_date,
       ub.your_bid,
       COALESCE(mb.highest_bid, 0) AS highest_bid,
       mb.highest_bidder
FROM item i
JOIN UserBids ub ON i.id = ub.item_id
LEFT JOIN MaxBids mb ON i.id = mb.item_id
ORDER BY i.bid_closing_date DESC
```

Purpose:

Retrieve the status of the current user's bids.

Parameters:

- `bidder_id`: ID of the user placing the bids (automatically set by the current user).
- Return List of items with their bid status.

Handling:

- If the query returns a result, the user's bidding status is displayed.
- If the query returns an empty result, the user has no active bids by the current user.

4.6 Check the account (sold and purchased items)

4.6.1 View the sold and purchased items for the current user

```sql
-- retrieve sold items
SELECT item.category,
       item.id AS item_id,
       billing.purchase_date AS sold_date,
       billing.amount_due_buyers_need_to_pay AS sold_price,
       billing.buyer_id,
       CAST(billing.commission_fee AS VARCHAR) AS commission_fee
FROM billing
JOIN item ON billing.sold_item_id = item.id
WHERE billing.seller_id = ?


-- retrieve purchased items
SELECT item.category,
       item.id AS item_id,
       billing.purchase_date AS purchased_date,
       CAST(billing.amount_due_buyers_need_to_pay AS VARCHAR) AS purchased_price,
       billing.seller_id
FROM billing
JOIN item i ON billing.sold_item_id = item.id
WHERE billing.buyer_id = ?
```

Purpose:

Retrieve the account details of the current user, including sold and purchased items.

Parameters:

- `user_id`: ID of the user to retrieve account details for (automatically set by the current user).
- Return List of sold and purchased items with their details.

Handling:

- If the query returns a result, the user's account details are displayed.
- If the query returns an empty result, the user has no transaction history.

4.7 Admin features

4.7.1 View Sold items per category

```
SELECT i.description,
       b.purchase_date,
       b.seller_id,
       b.buyer_id,
       b.amount_due_buyers_need_to_pay,
       b.commission_fee
FROM billing b
JOIN item i ON b.sold_item_id = i.id
WHERE i.category = ?
```

Purpose:

Retrieve sold items based on the specified category.

Parameters:

- `category`: Type of item to filter by.
- Return List of sold items in the specified category.

Handling:

- If the query returns a result, the sold items are displayed.
- If the query returns an empty result, there are no sold items in the specified category.
- If the category is not valid, the query not executed. (due to the code implementation)

4.7.2 View account balance for specific user

```
SELECT i.description,
       b.purchase_date,
       b.buyer_id,
       b.amount_due_buyers_need_to_pay,
       b.commission_fee
FROM billing b
JOIN item i ON b.sold_item_id = i.id
WHERE b.seller_id = ?
```

Purpose:

Retrieve the account balance for specific user based on the total profit. (sold items)

Parameters:

- `seller_id`: ID of the user to retrieve account balance for.
- Return List of users with their total profit. (sum of sold items bid price - sum of commission fee)

Handling:

- If the query returns a result, the user's total profit is displayed.
- If the query returns an empty result, there are no transactions for the specified user.
- If the seller ID is not valid, the query will fail.

4.7.3 View the seller ranking with the total profit

```
SELECT seller_id,
       COUNT(*) AS items_sold,
       SUM(amount_of_money_sellers_need_to_get_paid) AS total_profit
FROM billing
GROUP BY seller_id
ORDER BY total_profit DESC
```

Purpose:

Retrieve the seller ranking based on the total profit.

Return:

List of sellers with their total profit and number of items sold.

Handling:

- If the query returns a result, the seller ranking is displayed.
- If the query returns an empty result, there are no transactions to rank.

4.7.4 View the buyer ranking with the total purchase amount

```
SELECT buyer_id,
       COUNT(*) AS items_purchased,
       SUM(amount_due_buyers_need_to_pay) AS total_spent
FROM billing
GROUP BY buyer_id
ORDER BY total_spent DESC
```

Purpose:

Retrieve the buyer ranking based on the total purchase amount.

Return:

List of buyers with their total purchase amount and number of items purchased.

Handling:

- If the query returns a result, the buyer ranking is displayed.
- If the query returns an empty result, there are no transactions to rank.

4.8 Advance Processing of expired items (bid closing date passed)

NOTE This is pre-executed whenever the item qeurying is needed.

- print sold items per category
- print account (normal user menu, admin menu)
- print seller/buyer ranking
- check sell status
- check bid status
- buy item (this case not executed because excluding expired items are already included in item list query)

```
SELECT i.id,
       i.seller_id,
       b.bidder_id AS winner_id,
       b.bid_price AS winning_bid
```

```sql
FROM item i
LEFT JOIN (
    SELECT DISTINCT ON (item_id) item_id,
           bidder_id,
           bid_price
    FROM bid
    ORDER BY item_id, bid_price DESC, date_posted ASC
) b ON i.id = b.item_id
WHERE i.status = 'AVAILABLE'
  AND i.bid_closing_date < CURRENT_TIMESTAMP
  AND i.id NOT IN (SELECT sold_item_id FROM billing)

INSERT INTO billing (sold_item_id, seller_id, buyer_id, amount_due_buyers_need_to_pay)
VALUES (?, ?, ?, ?)

UPDATE item
SET status = ?
WHERE id = ?
```

Purpose:

Retrieve all expired items that not in the billing table and update the status, add billing information to the billing table.

Parameters:

- `item_id`: ID of the item that has expired.
- `seller_id`: ID of the seller of the item.
- `winner_id`: ID of the winning bidder.
- `winning_bid`: Winning bid amount.
- `amount_due_buyers_need_to_pay`: Amount due from the buyer.
- Return List of expired items that need to be processed.
- `status`: New status of the item (`SOLD`).
- `item_id`: ID of the item to update.

Handling:

- If the query returns a result, the expired items are processed.
- If the query returns an empty result, there are no expired items to process.