

Corporate State University Baden-Württemberg Mannheim

## **Database Project**

### **BidBits**

**Study program Business Informatics**

**Spezialisierung Data Science**

Authors:	David Schäfer, Eric Echtermeyer
Student numbers:	999999, 6373947
Course:	WWI-21-DSA
Director of studies:	Prof. Dr.-Ing. habil. Dennis Pfisterer
Lecturer:	Petko Rutesic

# Contents

<b>1</b>	<b>Database specification</b>	<b>1</b>
1.1	Tables . . . . .	2
1.2	Relationships . . . . .	5
1.3	Views . . . . .	6
1.4	Stored Procedure . . . . .	9
<b>2</b>	<b>Normalization Analysis</b>	<b>11</b>
<b>3</b>	<b>SQL Queries and Indices</b>	<b>13</b>

# 1 Database specification

In the subsequent section, the specifics of the database will be explained. Figure 1.1 showcases the Entity-Relationship Diagram (ERD) for the database schema which visually represents the structure of the implemented database. In the following, detailed definitions for used tables, relationships, views and stored procedures will be explained too.

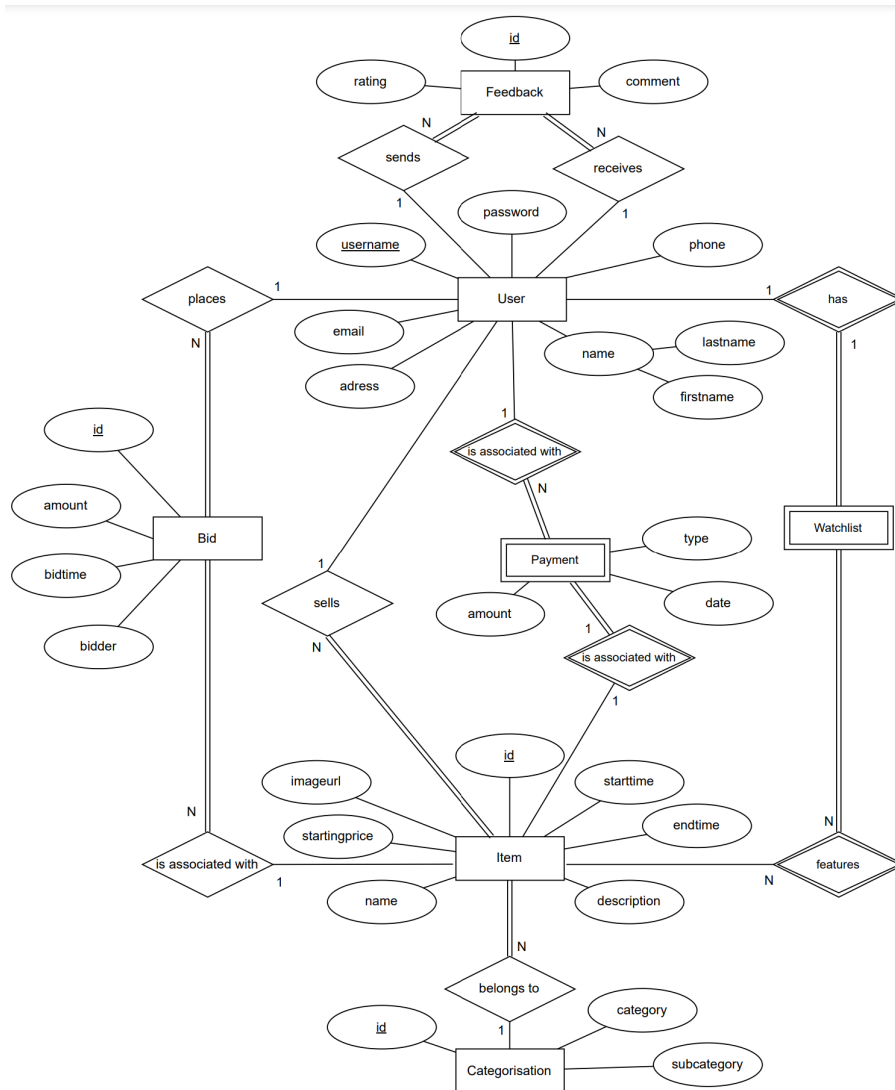


Figure 1.1: Entity-Relationship Diagram

## 1.1 Tables

The user table is designed to store all details about the users of the auction system. This includes columns such as 'username', which serves as the primary key and a unique identifier for each user, 'email', the user's unique email address, 'password', the user's hashed password for enhanced security, 'firstName' and 'lastName', which are the first and last name of the user respectively, 'name', a composite of the first and last names, 'address', the user's physical address, and 'phone', the unique phone number of the user. All attributes must have values to create a new user entry.

```
1      CREATE TABLE "user" (  
2          username VARCHAR(255) PRIMARY KEY,  
3          email EMAIL NOT NULL UNIQUE,  
4          password VARCHAR(255) NOT NULL,  
5          firstName VARCHAR(50) NOT NULL,  
6          lastName VARCHAR(50) NOT NULL,  
7          name VARCHAR(100) GENERATED ALWAYS AS (firstName || '␣'  
            ↪ || lastName) STORED,  
8          address TEXT NOT NULL,  
9          phone VARCHAR(20) NOT NULL UNIQUE  
10     );
```

The item table is responsible for tracking all items listed for auction. It comprises columns such as 'id', a unique identifier for each item and the primary key for this table, 'name', the item's name, 'description', a detailed description of the item, 'startingPrice', the initial price for the auction of this item, 'startTime' and 'endTime', the timestamps for the beginning and end of the auction, 'imageUrl', the URL for an image of the item, 'user\_username', the username of the user who listed the item, a foreign key referencing the user table, and 'category\_id', the id of the item's category, a foreign key referencing the categorisation table.

```
1      CREATE TABLE Item (  
2          id INTEGER DEFAULT nextval('item_sequence') PRIMARY KEY,  
3          name VARCHAR(50) NOT NULL,  
4          description TEXT NOT NULL,  
5          startingPrice NUMERIC DEFAULT 0,  
6          startTime TIMESTAMP NOT NULL,
```

```
7         endTime TIMESTAMP NOT NULL,  
8         imageUrl TEXT NOT NULL,  
9         user_username VARCHAR(50) REFERENCES "user"(username)  
10    ON DELETE SET NULL,  
11         category_id INTEGER REFERENCES Categorisation(id)  
12    );
```

The categorisation table categorizes the items being auctioned. It consists of three columns - 'id', 'category', and 'subcategory'. 'id' is a unique identifier for each category or subcategory pairing and serves as the primary key for this table. 'category' and 'subcategory' are text-based columns that describe the item category and the corresponding subcategory.

```
1    CREATE TABLE Categorisation (  
2        id SERIAL PRIMARY KEY,  
3        category VARCHAR(50) NOT NULL,  
4        subcategory VARCHAR(50) NOT NULL  
5    );
```

The bid table contains information regarding all bids placed on the items. It includes 'id', a unique identifier for each bid and the primary key for this table, 'amount', the bid's amount, 'bidTime', the timestamp when the bid was placed, 'user\_username', the username of the user who placed the bid, a foreign key referencing the user table, and 'item\_id', the id of the item being bid on, a foreign key referencing the item table.

```
1    CREATE TABLE Bid (  
2        id SERIAL PRIMARY KEY,  
3        amount NUMERIC NOT NULL,  
4        bidTime TIMESTAMP NOT NULL,  
5        user_username VARCHAR(50) REFERENCES "user"(username)  
6    ON DELETE SET NULL,  
7        item_id INTEGER REFERENCES Item(id)  
8    );
```

The watchlist table allows users to track items they are interested in. It includes 'user\_username', the username of the user interested in an item, a foreign key referencing the user table, and 'item\_id', the id of the item the user is interested in, a foreign key referencing the item

table. The combination of 'user\_username' and 'item\_id' is unique, preventing duplicate entries in the watchlist.

```
1      CREATE TABLE Watchlist (  
2          user_username VARCHAR(50) REFERENCES "user"(username)  
3      ON DELETE CASCADE,  
4          item_id SERIAL REFERENCES Item(id),  
5      CONSTRAINT unique_watchlist_entry UNIQUE (user_username,  
6          ↪ item_id)  
7      );
```

The feedback table facilitates users in leaving feedback about their transactions. It comprises 'feedbackID', a unique identifier for each feedback and the primary key for this table, 'rating', the rating given by the sender to the receiver on a scale from 0 to 10, 'comment', a text field for additional comments, 'sender' and 'receiver', the usernames of the user giving and receiving the feedback, both foreign keys referencing the user table.

```
1      CREATE TABLE Feedback (  
2          feedbackID SERIAL PRIMARY KEY,  
3          rating INTEGER NOT NULL CHECK (rating BETWEEN 0 AND 10),  
4          comment TEXT,  
5          sender VARCHAR(50) REFERENCES "user"(username)  
6      ON DELETE SET NULL,  
7          receiver VARCHAR(50) REFERENCES "user"(username)  
8      ON DELETE SET NULL  
9      );
```

The payment table records all payments made for the items. It includes 'amount', the amount paid, 'date', the timestamp when the payment was made, 'type', the method of payment, 'user\_username', the username of the user making the payment, a foreign key referencing the user table, and 'item\_id', the id of the item for which the payment was made, a foreign key referencing the item table.

```
1      CREATE TABLE Payment (  
2          amount NUMERIC NOT NULL,  
3          date TIMESTAMP NOT NULL,  
4          type PAYMENT_TYPE NOT NULL,  
5          user_username VARCHAR(50) REFERENCES "user"(username)
```

```
6         ON DELETE SET NULL,  
7         item_id INTEGER REFERENCES Item(id)  
8     );
```

## 1.2 Relationships

**User Table - Item Table:** Each user can list many items for auction, but each item is listed by only one user. The relationship is established via the user\_username foreign key in the Item table.

**User Table - Bid Table:** Each user can place many bids, but each bid is placed by only one user. This relationship is established via the user\_username foreign key in the Bid table.

**User Table - Watchlist Table:** Each user can have many items on their watchlist, but each watchlist entry is linked to only one user. This relationship is established via the user\_username foreign key in the Watchlist table.

**User Table - Feedback Table:** Each user can both give and receive feedback many times, but each feedback entry is given by and given to only one user. This relationship is established via the sender and receiver foreign keys in the Feedback table.

**User Table - Payment Table:** Each user can make many payments, but each payment is made by only one user. This relationship is established via the user\_username foreign key in the Payment table.

**Item Table - Bid Table:** Each item can have many bids, but each bid is placed on only one item. The relationship is established via the item\_id foreign key in the Bid table.

**Item Table - Watchlist Table:** Each item can be on the watchlist of many users, but each watchlist entry is for only one item. This relationship is established via the item\_id foreign key in the Watchlist table.

**Item Table - Payment Table:** Each item has one payment associated with it, and each payment is linked to only one item. This relationship is established via the item\_id foreign key in the Payment table.

**Categorisation Table - Item Table:** Each category can have many items, but each item belongs to only one category. The relationship is established through the category\_id foreign key in the Item table.

## 1.3 Views

The database also consists of two views, item\_status and user\_statistics. The items\_status view is created to show the current status of each item in the auction. It contains various fields that provide important information about the item, its current highest bid, and the related users (highest bidder and seller). The purpose of this view is to aggregate relevant data from multiple tables and present it in a more readable and understandable format. It can be used to display the current state of items in a user-friendly way on an application or website.

The view is created by joining the item table with a subquery that calculates the highest bid for each item. The result is then joined with the bid table to retrieve the highest bidder's username. The selected fields in this view include the item's name, id, description, image URL, remaining time until the auction ends, the highest bid, the highest bidder's username, and the seller's username.

```
1      CREATE VIEW items_status AS
2      SELECT
3          item.name AS title,
4          item.id AS item_id,
5          item.description,
6          item.imageUrl AS image_path,
7          EXTRACT(DAY FROM AGE(item.endtime, CURRENT_TIMESTAMP))
8              ⇨ AS time_left,
9          max_bids.highest_bid,
10         bid.user_username AS highest_bidder,
11         item.user_username AS seller
12 FROM item
13 JOIN (SELECT item_id, MAX(amount) AS highest_bid FROM bid GROUP
14         ⇨ BY item_id) AS max_bids
15 ON item.id = max_bids.item_id
16 JOIN bid
```



```
15         ON bid.item_id = max_bids.item_id
16         AND bid.amount = max_bids.highest_bid;
```

The user\_statistics materialized view in the database provides an aggregated summary of user activities on the auction platform. Being a materialized view, it stores the result of a complex and potentially time-consuming query, which can be refreshed as required, leading to faster data retrieval.

Each row in this view represents a unique user on the platform. The view contains a variety of fields calculated from different tables to give a comprehensive overview of each user's auction activities. The participated\_auctions field shows the total number of unique auctions where the user has placed a bid, as determined from the bid table. Another field, won\_auctions, displays the number of auctions that the user has won. It's calculated from the bid table, where the user's bid was the highest and the auction time had ended. The average\_rating field indicates the mean feedback rating received by the user. It's derived from the feedback table by averaging all the ratings received by a user. To understand the financial aspect, two more fields are included: total\_expenses and total\_income. total\_expenses shows the sum of all the payments made by a user for won auctions, calculated from the payment table. On the other hand, total\_income calculates the total amount received by a user for the items they have sold. It's calculated by joining the payment and item tables and summing up the amounts where the user is the seller.

```
1      CREATE MATERIALIZED VIEW user_statistics AS
2      SELECT
3          u.username,
4          COALESCE(participated_auctions.participated_auctions, 0)
           ↪ AS participated_auctions,
5          COALESCE(won_auctions.won_auctions, 0) AS won_auctions,
6          COALESCE(CAST(average_feedback.average_rating AS INTEGER
           ↪ ), 0) AS average_rating,
7          COALESCE(total_expenses.total_expenses, 0) AS
           ↪ total_expenses,
8          COALESCE(total_income.total_income, 0) AS total_income
9      FROM "user" u
10     LEFT JOIN (
11         SELECT user_username, COUNT(DISTINCT item_id) AS
           ↪ participated_auctions
```

```
12     FROM bid
13     GROUP BY user_username
14 ) AS participated_auctions ON u.username =
    ↳ participated_auctions.user_username
15 LEFT JOIN (
16 SELECT user_username, COUNT(DISTINCT bid.item_id) AS
    ↳ won_auctions
17 FROM bid
18 JOIN items_status ON bid.item_id = items_status.item_id AND bid
    ↳ .amount = items_status.highest_bid
19 WHERE items_status.time_left < 0
20 GROUP BY user_username
21 ) AS won_auctions ON u.username = won_auctions.user_username
22 LEFT JOIN (
23 SELECT receiver, AVG(rating) AS average_rating
24 FROM feedback
25 GROUP BY receiver
26 ) AS average_feedback ON u.username = average_feedback.receiver
27 LEFT JOIN (
28 SELECT user_username AS buyer, SUM(amount) AS total_expenses
29 FROM payment
30 GROUP BY user_username
31 ) AS total_expenses ON u.username = total_expenses.buyer
32 LEFT JOIN (
33 SELECT item.user_username AS seller, SUM(amount) AS
    ↳ total_income
34 FROM payment
35 JOIN item ON item.id = payment.item_id
36 GROUP BY item.user_username
37 ) AS total_income ON u.username = total_income.seller;
```

## 1.4 Stored Procedure

The purpose of the implemented stored procedure is to add a random item to the watchlist of a user when the user account is created. The item must be currently on auction, meaning its end time must be in the future.

A variable `random_item_id` of type integer is declared to store the id of a randomly selected item. It executes a `SELECT` query to get the id of a random item that is currently on auction. This is done by ordering the items from the `items_status` view (which contains the items with their current status) randomly and selecting the first one. Once it has the id of a random item, it inserts a new row into the watchlist table with the username of the new user and the id of the randomly selected item.

This stored procedure is run by a trigger named `user_created_trigger`. A trigger is a special type of stored procedure that runs automatically when an event associated with a table occurs such as insert, update, or delete. In this case, the trigger is configured to run the `add_random_item_to_watchlist()` stored procedure each time a new row is inserted into the user table. This is specified by the `AFTER INSERT ON "user"` part of the trigger creation statement.

```
1      CREATE OR REPLACE FUNCTION add_random_item_to_watchlist()
2      RETURNS TRIGGER AS $$
3      DECLARE
4          random_item_id INTEGER;
5      BEGIN
6
7          -- Get the id of a random item which is currently on auction
8          SELECT item_id
9          FROM items_status
10         WHERE time_left > 0
11         ORDER BY RANDOM()
12         LIMIT 1
13         INTO random_item_id;
14
15         -- Put that item on the watchlist of the newly created user
16         INSERT INTO watchlist (user_username, item_id)
17         VALUES (NEW.username, random_item_id);
```

```
18
19     RETURN NEW;
20     END;
21
22 $$ LANGUAGE plpgsql;
23
24 -- Create a trigger that executes the stored procedure
25     ↪ whenever a new entry is added to the user table
26 CREATE TRIGGER user_created_trigger
27 AFTER INSERT ON "user"
28 FOR EACH ROW
EXECUTE FUNCTION add_random_item_to_watchlist();
```

## 2 Normalization Analysis

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normal forms are a set of conditions or rules that a relation (table) in a database must adhere to, to qualify as a 'good' structure. This analysis will check whether the relations in the database meet at least the Third Normal Form (3NF).

Normal Form	Definition
First Normal Form (1NF)	A relation is in 1NF if it contains an atomic value for each attribute (column) in a record (row). It should also have a primary key that uniquely identifies each record.
Second Normal Form (2NF)	A relation is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. This essentially means there is no partial dependency of any column on the primary key.
Third Normal Form (3NF)	A relation is in 3NF if it is in 2NF and no non-key attribute is transitively dependent on the primary key.

Table 2.1: Definitions of Normal Forms

**1. Categorisation Table** The Categorisation table consists of three attributes: id, category, and subcategory. The primary key is 'id', and every 'id' refers to a unique category and subcategory. There are no duplicate or redundant data, and every attribute is atomic, which satisfies 1NF. Since there's only one candidate key (id), and all non-key attributes (category and subcategory) are fully dependent on it, 2NF is satisfied. There's no transitive dependency as there's only one candidate key, and thus the table also satisfies 3NF.

**2. User Table** The 'user' table has eight attributes: username, email, password, firstName, lastName, name, address, and phone. The primary key is 'username', and every 'username' refers to a unique user record. However, the 'name' attribute is a derived attribute (it's generated always as the concatenation of 'firstName' and 'lastName'). This breaks the rule of 1NF as it introduces redundancy. To satisfy 1NF, we could remove the 'name' attribute and derive it in our queries when needed. Assuming we consider the 'name' attribute as not violating 1NF, the table would meet the conditions for 2NF as there's only one candidate key, and all non-key attributes are fully dependent on it. For

3NF, the table doesn't have any non-key attribute that is transitively dependent on the primary key, so it satisfies 3NF as well.

**3. Item Table** The Item table has nine attributes: id, name, description, startingPrice, startTime, endTime, imageUrl, user\_username, and category\_id. The primary key is 'id', and every 'id' refers to a unique item. The table is in 1NF as all attributes are atomic, and each record is unique. The table is in 2NF as all non-key attributes are fully dependent on the primary key. The table is in 3NF as there are no transitive dependencies.

**4. Bid Table** The Bid table is in 1NF, 2NF, and 3NF. All attributes are atomic, each record is uniquely identified by 'id', all non-key attributes are fully dependent on the primary key, and there are no transitive dependencies.

**5. Watchlist Table** The Watchlist table has two attributes: user\_username and item\_id. The primary key is a combination of 'user\_username' and 'item\_id', and every combination refers to a unique watchlist entry. The table is in 1NF as all attributes are atomic, and each record is unique. The table is in 2NF as there are no non-key attributes, so the condition of full functional dependency is trivially satisfied. The table is also in 3NF as there are no transitive dependencies.

All tables in the given SQL code are in the Third Normal Form (3NF), assuming that the 'name' attribute in the 'user' table doesn't violate the 1NF. If it does, the 'user' table should be modified by removing the 'name' attribute to meet 1NF, 2NF, and 3NF.

## **3 SQL Queries and Indices**