ebd

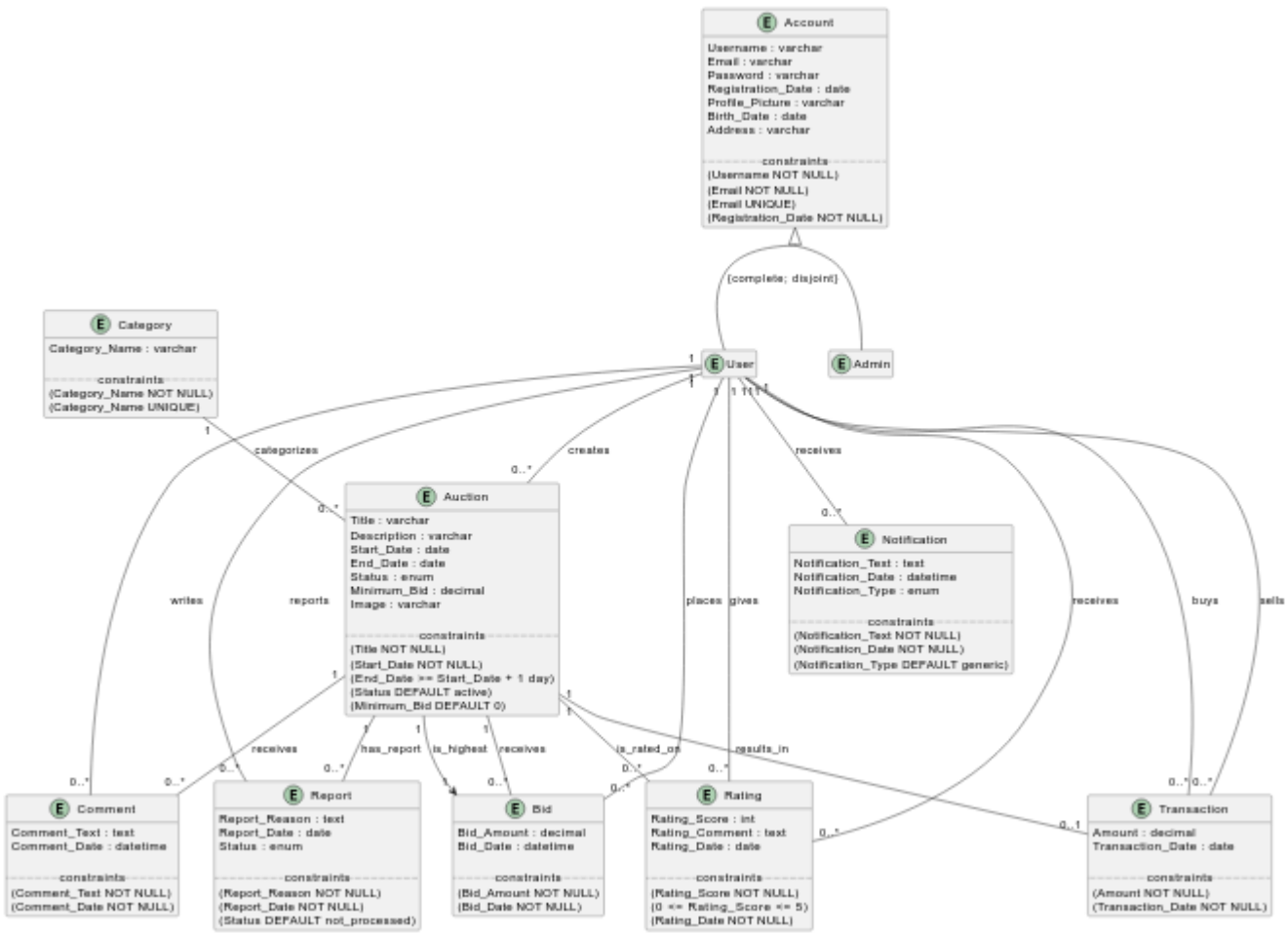
Last edited by **Alexandre Silva** 1 minute ago

EBD: Database Specification Component

A4: Conceptual Data Model

This artifact describes the relevant entities and relationships in the AuctionPeer database system.

1. Class diagram



2. Additional Business Rules

Rule ID	Description
BR.102	An auction can only be cancelled if there are no bids.
BR.103	A user cannot bid if his bid is the current highest.
BR.104	When a bid is made in the last 15 minutes of the auction, the deadline is extended by 30 minutes.
BR.105	A user cannot review their own auctions/account so as to keep reviews as unbiased as possible.
BR.106	When a user account is deleted, all of its activity history (auctions created, bids, etc.) will remain visible but its profile will be unrecognizable, i.e. any two deleted accounts are indistinguishable.
BR.107	An auction's start date must be lesser than its end date by at least a day.

A5: Relational Schema, validation and schema refinement

This artifact presents the Relational Schema derived from the mapping of the Conceptual Data Model. The Relational Schema specifies each relation schema, including attributes, domains, primary keys, foreign keys, and additional integrity constraints such as UNIQUE, DEFAULT, NOT NULL, and CHECK.

1. Relational Schema

Relation reference	Relation Compact Notation
R01	account(id, username **NN**, email **NN UK**, password **NN**, registration_date **NN**, profile_picture, birth_date, address)

Relation reference	Relation Compact Notation
R02	user(id -> Account, is_deleted)
R03	admin(id -> Account)
R04	auction(id, title **NN**, description, start_date **NN**, end_date **NN CK end_date >= start_date + 1 day**, status **DF active CK** status **IN** AuctionStatus, minimum_bid **CK DF 0**, current_bid -> bid **CK current_bid >= minimum_bid**, category_id -> Category, creator_id -> User, buyer_id -> User, picture)
R05	category(id, name **NN UK**)
R06	bid(id, amount **NN**, date **NN**, auctiono_id -> Auction, user_id -> User)
R07	rating(id, score **NN CK 0 <= score <= 5**, comment, date **NN**, auction_id -> Auction, rater_id -> User, reciever_id -> User)
R08	comment(id, text **NN**, date **NN**, auction_id -> Auction, user_id -> User)
R09	report(id, reason **NN**, date **NN**, status **DF not_processed CK** status **IN** ReportStatus, auction_id -> Auction, user_id -> User)
R10	notification(id, text **NN**, date **NN**, type **DF generic CK** type **IN** NotifType, reciever -> User)
R11	transaction(id, amount **NN**, date **NN**, auction_id -> Auction **NN**)

Legend:

- NN - Not Null
- UK - Unique
- DF - Default
- CK - Check

2. Domains

> The specification of additional domains can also be made in a compact form, using the notation:

Domain Name	Domain Specification
AuctionStatus	ENUM ('active', 'ended', 'canceled')
ReportStatus	ENUM ('not_processed', 'discarded', 'processed')
NotifType	ENUM ('new_bid', 'bid_surpassed', 'new_comment', 'report')

3. Schema validation

> To validate the Relational Schema obtained from the Conceptual Model, all functional dependencies are identified and the normalization of all relation schemas is accomplished.

Table R01: Account

TABLE R01	Account
Keys	{ Username }, { Email }
Functional Dependencies:	
FD0101	Username → {Email, Password, Registration_Date, Profile_Picture, Birth_Date, Address}
FD0102	Email → {Username, Password, Registration_Date, Profile_Picture, Birth_Date, Address}
NORMAL FORM	BCNF

Table R02: User

TABLE R02	User
Keys	{ Username }

TABLE R02	User
Functional Dependencies:	
FD0201	Username → {all inherited attributes from Account}
NORMAL FORM	BCNF

Table R03: Admin

TABLE R03	Admin
Keys	{ Username }
Functional Dependencies:	
FD0301	Username → {all inherited attributes from Account}
NORMAL FORM	BCNF

Table R04: Auction

TABLE R04	Auction
Keys	{ Auction_ID }
Functional Dependencies:	
FD0401	Auction_ID → {Title, Description, Start_Date, End_Date, Status, Minimum_Bid, Picture}
NORMAL FORM	BCNF

Table R05: Category

TABLE R05	Category
Keys	{ Category_Name }
Functional Dependencies:	
FD0501	Category_Name → {Category_Name} (trivial)
NORMAL FORM	BCNF

Table R06: Bid

TABLE R06	Bid
Keys	{ Bid_ID }, { Auction_ID, User_ID }
Functional Dependencies:	
FD0601	Bid_ID → {Bid_Amount, Bid_Date}
FD0602	Auction_ID, User_ID → {Bid_Amount, Bid_Date}
NORMAL FORM	BCNF

Table R07: Rating

TABLE R07	Rating
Keys	{ Rating_ID }, { Auction_ID, User_ID }
Functional Dependencies:	
FD0701	Rating_ID → {Rating_Score, Rating_Comment, Rating_Date}

TABLE R07	Rating
FD0702	Auction_ID, User_ID → {Rating_Score, Rating_Comment, Rating_Date}
NORMAL FORM	BCNF

Table R08: Comment

TABLE R08	Comment
Keys	{ Comment_ID }, { Auction_ID, User_ID }
Functional Dependencies:	
FD0801	Comment_ID → {Comment_Text, Comment_Date}
FD0802	Auction_ID, User_ID → {Comment_Text, Comment_Date}
NORMAL FORM	BCNF

Table R09: Report

TABLE R09	Report
Keys	{ Report_ID }, { Auction_ID, User_ID }
Functional Dependencies:	
FD0901	Report_ID → {Report_Reason, Report_Date, Status}
FD0902	Auction_ID, User_ID → {Report_Reason, Report_Date, Status}
NORMAL FORM	BCNF

Table R10: Notification

TABLE R10	Notification
Keys	{ Notification_ID }, { User_ID }
Functional Dependencies:	
FD1001	Notification_ID → {Notification_Text, Notification_Date, Notification_Type}
FD1002	User_ID → {Notification_Text, Notification_Date, Notification_Type}
NORMAL FORM	BCNF

Table R11: Transaction

TABLE R11	Transaction
Keys	{ Transaction_ID }, { Auction_ID, User_ID }
Functional Dependencies:	
FD1101	Transaction_ID → {Amount, Transaction_Date}
FD1102	Auction_ID, User_ID → {Amount, Transaction_Date}
NORMAL FORM	BCNF

Because all relations are in the Boyce–Codd Normal Form (BCNF), the relational schema is also in the BCNF and, therefore, the schema does not need to be further normalized.

A6: Indexes, triggers, transactions and database population

This artifact includes the schema of the database, detailing the identification and characterization of indexes, the implementation of data integrity

rules using triggers, and the definition of user-defined functions.

Additionally, it outlines the database transactions required to maintain data integrity during concurrent access, specifying and justifying the isolation level for each transaction.

The artifact also provides the database workload and the full database creation script, containing all SQL commands necessary to define integrity constraints, indexes, and triggers. Lastly, a separate script with INSERT statements for populating the database is included.

1. Database Workload

Table: Relation Data

Relation	Relation Name	Order of Magnitude	Estimated Growth
R01	account	10 k (tens of thousands)	10 (tens) / day
R02	user	10 k (tens of thousands)	10 (tens) / day
R03	admin	10 (tens)	Infrequent
R04	auction	1 k (thousands)	10 (tens) / day
R05	category	100 (hundreds)	Infrequent
R06	bid	10k (tens of thousands)	50-100 (tens to hundreds) / day
R07	rating	1k (thousands)	5-10 (units to tens) / day
R08	comment	10 k (thousands)	10-20 (tens) / day
R09	report	1k (thousands)	5 (units) / day
R10	notification	100 (hundreds of thousands) k	100 (hundreds) / day
R11	transaction	1 (tens of thousands) k	5-10 (units to tens) / day

2. Proposed Indices

2.1. Performance Indices

Index	**IDX01**
Relation	auction
Attribute	creator_id
Type	B-tree
Cardinality	High
Clustering	No
Justification	Frequently filtering auctions by creator_id can improve query performance for searches, especially if users often query their own or specific sellers’ auctions.
`SQL code`	CREATE INDEX IDX01 ON auction USING BTREE(creator_id);

Index	**IDX02**
Relation	bid
Attribute	auction_id
Type	B-tree
Cardinality	High
Clustering	Yes
Justification	Since bids are usually queried by auction_id to retrieve all bids for a specific auction, clustering bids by auction_id can reduce the disk I/O and optimize read performance.

Index	IDX02
`SQL code`	CREATE INDEX IDX02 ON bid USING BTREE(auction_id);

Index	IDX03
Relation	transaction
Attribute	auction_id
Type	B-tree
Cardinality	Medium
Clustering	No
Justification	Helps in fast retrieval of all transactions associated with specific auctions, optimizing for the likely use case of checking transaction history or order completion.
`SQL code`	CREATE INDEX IDX03 ON transactions USING BTREE(auction_id);

2.2. Full-text Search Indices

Index	IDX11
Relation	auction
Attribute	title, description
Type	GIN
Clustering	No
Justification	This index supports full-text search for auctions, allowing users to find auctions based on keywords in titles and descriptions. GIN indexing is ideal because titles and descriptions are typically stable once created, providing efficient keyword retrieval.
`SQL code`	

```
-- Auction Full-text Search Index
ALTER TABLE auction
ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION auction_search_update() RETURNS TRIGGER AS $$
BEGIN
 IF TG_OP = 'INSERT' OR (TG_OP = 'UPDATE' AND (NEW.title <> OLD.title OR NEW.description <> OLD.description)) THEN
     NEW.tsvectors = (
         setweight(to_tsvector('english', NEW.title), 'A') ||
         setweight(to_tsvector('english', NEW.description), 'B')
     );
 END IF;
 RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER auction_search_update
 BEFORE INSERT OR UPDATE ON auction
 FOR EACH ROW
 EXECUTE PROCEDURE auction_search_update();

CREATE INDEX auction_search_idx ON auction USING GIN (tsvectors);
```

3. Triggers

Trigger	TRIGGER01

Trigger	TRIGGER01
Description	An auction can only be cancelled if there are no bids.
Justification	Ensures that an auction cannot be cancelled if it has received bids, protecting user expectations and preserving the integrity of active bidding processes.
SQL code	

```
CREATE OR REPLACE FUNCTION check_bids_before_cancellation()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM bid WHERE auction_id = NEW.id) > 0 THEN
        RAISE EXCEPTION 'Auction with id % cannot be canceled because there are existing bids.', NEW.id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_auction_cancellation_with_bids
BEFORE UPDATE ON auction
FOR EACH ROW
WHEN (NEW.status = 'canceled')
EXECUTE FUNCTION check_bids_before_cancellation();
```

Trigger	TRIGGER02
Description	A user cannot bid if his bid is the current highest.
Justification	Prevents a user from repeatedly placing the highest bid, which could be seen as artificially inflating the price or obstructing fair bidding by other participants.
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_duplicate_highest_bid()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT user_id
        FROM bid
        WHERE auction_id = NEW.auction_id
        ORDER BY amount DESC, date DESC
        LIMIT 1) = NEW.user_id THEN
        RAISE EXCEPTION 'User % already has the highest bid on auction %.', NEW.user_id, NEW.auction_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_user_duplicate_highest_bid
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_highest_bid();
```

Trigger	TRIGGER03
Description	When a bid is made in the last 15 minutes of the auction, the deadline is extended by 30 minutes.
Justification	Extending the auction deadline when a bid is placed close to the end time helps to prevent last-second bids from unfairly ending the auction, giving other participants a fair chance to respond.
SQL code	

```
CREATE OR REPLACE FUNCTION extend_auction_if_bid_late()
```

```
RETURNS TRIGGER AS $$
BEGIN

    IF (NEW.date >= (SELECT end_date - INTERVAL '15 minutes'
                     FROM auction
                     WHERE id = NEW.auction_id)) THEN
        UPDATE auction
        SET end_date = end_date + INTERVAL '30 minutes'
        WHERE id = NEW.auction_id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER extend_auction_on_late_bid
AFTER INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION extend_auction_if_bid_late();
```

Trigger	TRIGGER04
Description	A user cannot review their own auctions/account so as to keep reviews as unbiased as possible.
Justification	Maintains the objectivity and trustworthiness of the platform's rating system by preventing users from rating their own auctions or accounts.
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_self_review()
RETURNS TRIGGER AS $$
BEGIN

    IF (SELECT creator_id FROM auction WHERE id = NEW.auction_id) = NEW.rater_id THEN
        RAISE EXCEPTION 'User cannot review their own auction (ID: %).', NEW.auction_id;
    END IF;
    IF NEW.receiver_id = NEW.rater_id THEN
        RAISE EXCEPTION 'User cannot review their own account (ID: %).', NEW.receiver_id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_user_self_review
BEFORE INSERT ON rating
FOR EACH ROW
EXECUTE FUNCTION prevent_self_review();
```

Trigger	TRIGGER05
Description	When a user account is deleted, anonymizing personal data ensures privacy and security while allowing the account's historical activities to remain in the system for reference and continuity.
SQL code	

```
CREATE OR REPLACE FUNCTION anonymize_user_account()
RETURNS TRIGGER AS $$
BEGIN

    UPDATE account
    SET
```

```
        username = 'deleted_user_' || NEW.id,
        email = 'deleted_' || NEW.id || '@example.com',
        password = NULL,  -- Clear the password for security
        profile_picture = NULL,
        address = NULL,
        birth_date = NULL
    WHERE id = NEW.id;


    UPDATE users
    SET is_deleted = TRUE
    WHERE id = NEW.id;


    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER anonymize_user_before_delete
BEFORE DELETE ON account
FOR EACH ROW
EXECUTE FUNCTION anonymize_user_account();
```

Trigger	TRIGGER06
Description	An auction's start date must be lesser than its end date by at least a day.
Justification	Enforces that each auction has a logical timeline, with the end date at least one day after the start date, ensuring users have adequate time to participate in the auction.
SQL code	

```
CREATE OR REPLACE FUNCTION check_auction_dates()
RETURNS TRIGGER AS $$
BEGIN

    IF NEW.end_date < NEW.start_date + INTERVAL '1 day' THEN
        RAISE EXCEPTION 'End date must be at least one day greater than start date for auction ID: %', NEW.id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_auction_date_constraints
BEFORE INSERT OR UPDATE ON auction
FOR EACH ROW
EXECUTE FUNCTION check_auction_dates();
```

Trigger	TRIGGER07
Description	Administrators cannot create auctions.
Justification	Ensures that only regular users can create auctions, maintaining role separation and preventing conflicts of interest.
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_admin_auction_creation()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM admin WHERE id = NEW.creator_id) THEN
        RAISE EXCEPTION 'Administrators are not allowed to create auctions.';
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_admin_auction_creation_trigger
BEFORE INSERT ON auction
FOR EACH ROW
EXECUTE FUNCTION prevent_admin_auction_creation();
```

Trigger	TRIGGER08
Description	Administrators cannot place bids.
Justification	Prevents administrators from participating in bidding, upholding the impartiality expected from administrative roles and ensuring fair competition among users.
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_admin_bid_placement()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM admin WHERE id = NEW.user_id) THEN
        RAISE EXCEPTION 'Administrators are not allowed to place bids.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_admin_bid_placement_trigger
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_admin_bid_placement();
```

Trigger	TRIGGER09
Description	An auction can only be cancelled if there are no bids.
Justification	Ensures that an auction cannot be canceled if it has received bids, protecting user expectations and preserving the integrity of active bidding processes.
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_auction_cancellation_with_bids()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM bid WHERE auction_id = NEW.id) > 0 THEN
        RAISE EXCEPTION 'Cannot cancel auction with existing bids.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_auction_cancellation_trigger
BEFORE UPDATE ON auction
FOR EACH ROW
WHEN (NEW.status = 'canceled')
EXECUTE FUNCTION prevent_auction_cancellation_with_bids();
```

Trigger	TRIGGER10
Description	A user cannot bid if his bid is the current highest.
Justification	Ensures that users cannot place a bid if they already hold the highest bid, preventing redundant bids and encouraging competitive bidding among different users.

Trigger	TRIGGER10
SQL code	

```
CREATE OR REPLACE FUNCTION prevent_duplicate_highest_bid()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM bid
        WHERE auction_id = NEW.auction_id
          AND user_id = NEW.user_id
          AND amount = (SELECT MAX(amount) FROM bid WHERE auction_id = NEW.auction_id)
    ) THEN
        RAISE EXCEPTION 'You cannot place a bid if you already have the highest bid.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_duplicate_highest_bid_trigger
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_highest_bid();
```

Trigger	TRIGGER11
Description	When a bid is made in the last 15 minutes of the auction, the deadline is extended by 30 minutes.
Justification	Extends the auction deadline if a bid is placed close to the original end time, allowing more competition and ensuring interested bidders have a fair chance to participate.

| | SQL code |

```
CREATE OR REPLACE FUNCTION extend_auction_deadline()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT end_date FROM auction WHERE id = NEW.auction_id) - NEW.date <= INTERVAL '15 minutes' THEN
        UPDATE auction
        SET end_date = end_date + INTERVAL '30 minutes'
        WHERE id = NEW.auction_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER extend_auction_deadline_trigger
AFTER INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION extend_auction_deadline();
```

Trigger	TRIGGER12
Description	A user cannot review their own auctions/account so as to keep reviews as unbiased as possible.
Justification	Prevents users from reviewing their own auctions or accounts, ensuring that reviews remain unbiased and reflect the experiences of other users.

| | SQL code |

```
CREATE OR REPLACE FUNCTION prevent_self_review()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
```

```
        SELECT 1
        FROM auction
        WHERE id = NEW.auction_id
          AND creator_id = NEW.receiver_id
    ) THEN
        RAISE EXCEPTION 'You cannot review your own auction.';
    END IF;


    IF NEW.rater_id = NEW.receiver_id THEN
        RAISE EXCEPTION 'You cannot review your own account.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_self_review_trigger
BEFORE INSERT ON rating
FOR EACH ROW
EXECUTE FUNCTION prevent_self_review();
```

Trigger	TRIGGER13
Description	When a user account is deleted, all of its activity history (auctions created, bids, etc.) will remain visible but its profile will be unrecognizable, i.e. any two deleted accounts are indistinguishable.
Justification	Ensures that when a user account is deleted, their identifiable information is anonymized while retaining the visibility of their activity history, preserving the integrity of the data while protecting user privacy.

| | **SQL code** |

```
CREATE OR REPLACE FUNCTION anonymize_user_data()
RETURNS TRIGGER AS $$
BEGIN

    UPDATE account
    SET username = 'deleted_user_' || NEW.id,
        email = 'deleted_user_' || NEW.id || '@example.com',
        password = 'deleted',
        profile_picture = NULL,
        birth_date = NULL,
        address = NULL
    WHERE id = OLD.id;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER anonymize_user_trigger
AFTER DELETE ON users
FOR EACH ROW
EXECUTE FUNCTION anonymize_user_data();
```

Trigger	TRIGGER14
Description	An auction's start date must be lesser than its end date by at least a day.
Justification	Ensures that the auction’s start date is always at least one day earlier than the end date, maintaining clear timelines and user expectations for auction events.

| | **SQL code** |

```
CREATE OR REPLACE FUNCTION check_auction_dates()
RETURNS TRIGGER AS $$
```

```
BEGIN
    IF NEW.end_date <= NEW.start_date + INTERVAL '1 day' THEN
        RAISE EXCEPTION 'The auction end date must be at least one day after the start date.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_auction_dates_trigger
BEFORE INSERT OR UPDATE ON auction
FOR EACH ROW
EXECUTE FUNCTION check_auction_dates();
```

4. Transactions

> Transactions needed to assure the integrity of the data.

Transaction	TRAN01
Description	Fetch bids on a specified auction.
Justification	Ensures there that no bids or updates to the auction occur during the transaction, protecting the retrieval of bids from phantom reading. Its isolation level is READ ONLY as there are only SELECTs.
Isolation level	SERIALIZABLE READ ONLY

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY;

SELECT COUNT(*)
FROM bid
WHERE auction_id = $auction_id;

SELECT bid.amount, bid.date, account.username AS bidder
FROM bid
INNER JOIN account ON account.id = bid.user_id
WHERE bid.auction_id = $auction_id
ORDER BY bid.amount DESC;

END TRANSACTION;
```

Transaction	TRAN02
Description	Fetch notifications for a specified user.
Justification	Ensures consistent view of notifications for a user without updates occurring during the transaction. READ ONLY because it is composed of SELECTs.
Isolation level	SERIALIZABLE READ ONLY

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY;

SELECT COUNT(*)
```

```
FROM notifications
WHERE receiver_id = $user_id;

SELECT notifications.text, notifications.date, notifications.type
FROM notifications
WHERE receiver_id = $user_id
ORDER BY notifications.date DESC;

END TRANSACTION;
```

Transaction	TRAN03
Description	Insert a bid and update the current top bid in an auction;
Justification	Ensures that, when a new bid is placed, the bid is recorded and the auction's current highest bid is also updated. Using the REPEATABLE READ isolation level prevents concurrent transactions from affecting the current bid information, making sure that no transaction reads an outdated highest bid amount.
Isolation level	REPEATABLE READ

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

INSERT INTO bid (amount, date, auction_id, user_id)
VALUES ($amount, $date, $auction_id, $user_id);

UPDATE auction
SET current_bid = $amount
WHERE id = $auction_id AND (current_bid IS NULL OR current_bid < $amount);

END TRANSACTION;
```

Transaction	TRAN04
Description	Delete a user and associated records.
Justification	Ensures that when a user is deleted, all associated records in 'bids', 'ratings', 'comments', 'notifications' and other related tables are also removed to avoid orphaned data. Using the SERIALIZABLE isolation level to prevent other transactions from affecting these deletions while in progress.
Isolation level	SERIALIZABLE

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

DELETE FROM bids WHERE user_id = $user_id;
DELETE FROM ratings WHERE rater_id = $user_id OR receiver_id = $user_id;
DELETE FROM comments WHERE user_id = $user_id;
DELETE FROM notifications WHERE receiver_id = $user_id;
DELETE FROM reports WHERE user_id = $user_id;
DELETE FROM users WHERE id = $user_id;
DELETE FROM accounts WHERE id = $user_id;

END TRANSACTION;
```

Transaction	TRAN05
Description	Fetch reviews for a specific user;
Justification	Ensures consistent view of reviews for a user without updates occurring during the transaction. READ ONLY because it is composed of SELECTs.
Isolation level	SERIALIZABLE READ ONLY

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY;

SELECT COUNT(*)
FROM ratings
WHERE rater_id = $user_id;

SELECT ratings.score, ratings.comment, accounts.id, accounts.username, auctions.*
FROM ratings
INNER JOIN accounts ON accounts.id = ratings.rater_id
INNER JOIN auctions ON auctions.id = ratings.auction_id
WHERE ratings.receiver_id = $user_id

END TRANSACTION;
```

Annex A. SQL Code

A.1. Database schema

```
SET search_path TO lbaw2451;

DROP TABLE IF EXISTS account CASCADE;
DROP TABLE IF EXISTS users CASCADE;
DROP TABLE IF EXISTS admin CASCADE;
DROP TABLE IF EXISTS category CASCADE;
DROP TABLE IF EXISTS auction CASCADE;
DROP TABLE IF EXISTS bid CASCADE;
DROP TABLE IF EXISTS rating CASCADE;
DROP TABLE IF EXISTS comment CASCADE;
DROP TABLE IF EXISTS report CASCADE;
DROP TABLE IF EXISTS notification CASCADE;
DROP TABLE IF EXISTS transactions CASCADE;

DROP TYPE IF EXISTS auction_status;
DROP TYPE IF EXISTS report_status;
DROP TYPE IF EXISTS notif_type;

CREATE TYPE auction_status AS ENUM ('active', 'ended', 'canceled');
CREATE TYPE report_status AS ENUM ('not_processed', 'discarded', 'processed');
CREATE TYPE notif_type AS ENUM ('generic', 'new_bid', 'bid_surpassed', 'auction_end', 'new_comment', 'report');

CREATE TABLE account (
    id SERIAL PRIMARY KEY,
    username TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    registration_date TIMESTAMP NOT NULL,
    profile_picture TEXT,
    birth_date DATE,
    address TEXT
);
```

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY REFERENCES account(id),
    is_deleted BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE admin (
    id INTEGER PRIMARY KEY REFERENCES account(id)
);

CREATE TABLE category (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL UNIQUE
);

CREATE TABLE auction (
    id SERIAL PRIMARY KEY,
    title TEXT NOT NULL,
    description TEXT,
    start_date TIMESTAMP NOT NULL,
    end_date TIMESTAMP NOT NULL CHECK (end_date >= start_date + INTERVAL '1 day'),
    status auction_status DEFAULT 'active',
    minimum_bid NUMERIC CHECK (minimum_bid >= 0) DEFAULT 0,
    current_bid NUMERIC CHECK (current_bid >= minimum_bid),
    category_id INTEGER REFERENCES category(id),
    creator_id INTEGER REFERENCES users(id),
    buyer_id INTEGER REFERENCES users(id),
    picture TEXT
);

CREATE TABLE bid (
    id SERIAL PRIMARY KEY,
    amount NUMERIC NOT NULL,
    date TIMESTAMP NOT NULL,
    auction_id INTEGER REFERENCES auction(id),
    user_id INTEGER REFERENCES users(id)
);

CREATE TABLE rating (
    id SERIAL PRIMARY KEY,
    score INTEGER NOT NULL CHECK (score >= 0 AND score <= 5),
    comment TEXT,
    date TIMESTAMP NOT NULL,
    auction_id INTEGER REFERENCES auction(id),
    rater_id INTEGER REFERENCES users(id),
    receiver_id INTEGER REFERENCES users(id)
);

CREATE TABLE comment (
    id SERIAL PRIMARY KEY,
    text TEXT NOT NULL,
    date TIMESTAMP NOT NULL,
    auction_id INTEGER REFERENCES auction(id),
    user_id INTEGER REFERENCES users(id)
);

CREATE TABLE report (
    id SERIAL PRIMARY KEY,
    reason TEXT NOT NULL,
    date TIMESTAMP NOT NULL,
    status report_status DEFAULT 'not_processed',
    auction_id INTEGER REFERENCES auction(id),
    user_id INTEGER REFERENCES users(id)
);

CREATE TABLE notification (
```

```
    id SERIAL PRIMARY KEY,
    text TEXT NOT NULL,
    date TIMESTAMP NOT NULL,
    type notif_type DEFAULT 'generic',
    receiver_id INTEGER REFERENCES users(id)
);

CREATE TABLE transactions (
    id SERIAL PRIMARY KEY,
    amount NUMERIC NOT NULL,
    date TIMESTAMP NOT NULL,
    auction_id INTEGER NOT NULL REFERENCES auction(id)
);

CREATE INDEX IDX01 ON auction USING BTREE(creator_id);
CREATE INDEX IDX02 ON bid USING BTREE(auction_id);
CREATE INDEX IDX03 ON transactions USING BTREE(auction_id);

-- IDX11
ALTER TABLE auction
ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION auction_search_update() RETURNS TRIGGER AS $$
BEGIN
 IF TG_OP = 'INSERT' OR (TG_OP = 'UPDATE' AND (NEW.title <> OLD.title OR NEW.description <> OLD.description)) THEN
     NEW.tsvectors = (
         setweight(to_tsvector('english', NEW.title), 'A') ||
         setweight(to_tsvector('english', NEW.description), 'B')
     );
 END IF;
 RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER auction_search_update
 BEFORE INSERT OR UPDATE ON auction
 FOR EACH ROW
 EXECUTE PROCEDURE auction_search_update();

CREATE INDEX auction_search_idx ON auction USING GIN (tsvectors);

--TRIGGER01
CREATE OR REPLACE FUNCTION check_bids_before_cancellation()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM bid WHERE auction_id = NEW.id) > 0 THEN
        RAISE EXCEPTION 'Auction with id % cannot be canceled because there are existing bids.', NEW.id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_auction_cancellation_with_bids
BEFORE UPDATE ON auction
FOR EACH ROW
WHEN (NEW.status = 'canceled')
EXECUTE FUNCTION check_bids_before_cancellation();

--TRIGGER02
CREATE OR REPLACE FUNCTION prevent_duplicate_highest_bid()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT user_id
        FROM bid
        WHERE auction_id = NEW.auction_id
```

```
        ORDER BY amount DESC, date DESC
        LIMIT 1) = NEW.user_id THEN
        RAISE EXCEPTION 'User % already has the highest bid on auction %.', NEW.user_id, NEW.auction_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_user_duplicate_highest_bid
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_highest_bid();

--TRIGGER03
CREATE OR REPLACE FUNCTION extend_auction_if_bid_late()
RETURNS TRIGGER AS $$
BEGIN

    IF (NEW.date >= (SELECT end_date - INTERVAL '15 minutes'
                     FROM auction
                     WHERE id = NEW.auction_id)) THEN
        UPDATE auction
        SET end_date = end_date + INTERVAL '30 minutes'
        WHERE id = NEW.auction_id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER extend_auction_on_late_bid
AFTER INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION extend_auction_if_bid_late();

--TRIGGER04
CREATE OR REPLACE FUNCTION prevent_self_review()
RETURNS TRIGGER AS $$
BEGIN

    IF (SELECT creator_id FROM auction WHERE id = NEW.auction_id) = NEW.rater_id THEN
        RAISE EXCEPTION 'User cannot review their own auction (ID: %).', NEW.auction_id;
    END IF;
    IF NEW.receiver_id = NEW.rater_id THEN
        RAISE EXCEPTION 'User cannot review their own account (ID: %).', NEW.receiver_id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_user_self_review
BEFORE INSERT ON rating
FOR EACH ROW
EXECUTE FUNCTION prevent_self_review();

--TRIGGER05
CREATE OR REPLACE FUNCTION anonymize_user_account()
RETURNS TRIGGER AS $$
BEGIN

    UPDATE account
    SET
        username = 'deleted_user_' || NEW.id,
```

```
        email = 'deleted_' || NEW.id || '@example.com',
        password = NULL,  -- Clear the password for security
        profile_picture = NULL,
        address = NULL,
        birth_date = NULL
    WHERE id = NEW.id;


    UPDATE users
    SET is_deleted = TRUE
    WHERE id = NEW.id;


    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER anonymize_user_before_delete
BEFORE DELETE ON account
FOR EACH ROW
EXECUTE FUNCTION anonymize_user_account();
```

```
--TRIGGER06
CREATE OR REPLACE FUNCTION check_auction_dates()
RETURNS TRIGGER AS $$
BEGIN

    IF NEW.end_date < NEW.start_date + INTERVAL '1 day' THEN
        RAISE EXCEPTION 'End date must be at least one day greater than start date for auction ID: %', NEW.id;
    END IF;


    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_auction_date_constraints
BEFORE INSERT OR UPDATE ON auction
FOR EACH ROW
EXECUTE FUNCTION check_auction_dates();

--TRIGGER07
CREATE OR REPLACE FUNCTION prevent_admin_auction_creation()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM admin WHERE id = NEW.creator_id) THEN
        RAISE EXCEPTION 'Administrators are not allowed to create auctions.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_admin_auction_creation_trigger
BEFORE INSERT ON auction
FOR EACH ROW
EXECUTE FUNCTION prevent_admin_auction_creation();

--TRIGGER08
CREATE OR REPLACE FUNCTION prevent_admin_bid_placement()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM admin WHERE id = NEW.user_id) THEN
        RAISE EXCEPTION 'Administrators are not allowed to place bids.';
    END IF;
    RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_admin_bid_placement_trigger
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_admin_bid_placement();

--TRIGGER09
CREATE OR REPLACE FUNCTION prevent_auction_cancellation_with_bids()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM bid WHERE auction_id = NEW.id) > 0 THEN
        RAISE EXCEPTION 'Cannot cancel auction with existing bids.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_auction_cancellation_trigger
BEFORE UPDATE ON auction
FOR EACH ROW
WHEN (NEW.status = 'canceled')
EXECUTE FUNCTION prevent_auction_cancellation_with_bids();

--TRIGGER10
CREATE OR REPLACE FUNCTION prevent_duplicate_highest_bid()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM bid
        WHERE auction_id = NEW.auction_id
          AND user_id = NEW.user_id
          AND amount = (SELECT MAX(amount) FROM bid WHERE auction_id = NEW.auction_id)
    ) THEN
        RAISE EXCEPTION 'You cannot place a bid if you already have the highest bid.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_duplicate_highest_bid_trigger
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_highest_bid();

--TRIGGER11
CREATE OR REPLACE FUNCTION extend_auction_deadline()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT end_date FROM auction WHERE id = NEW.auction_id) - NEW.date <= INTERVAL '15 minutes' THEN
        UPDATE auction
        SET end_date = end_date + INTERVAL '30 minutes'
        WHERE id = NEW.auction_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER extend_auction_deadline_trigger
AFTER INSERT ON bid
FOR EACH ROW
EXECUTE FUNCTION extend_auction_deadline();

--TRIGGER12
CREATE OR REPLACE FUNCTION prevent_self_review()
```

```
RETURNS TRIGGER AS $$
BEGIN
    -- Ensure the receiver is the creator of the auction
    IF NOT EXISTS (
        SELECT 1
        FROM auction
        WHERE id = NEW.auction_id
          AND creator_id = NEW.receiver_id
    ) THEN
        RAISE EXCEPTION 'The receiver must be the creator of the auction.';
    END IF;

    -- Ensure the rater is not the same as the receiver
    IF NEW.rater_id = NEW.receiver_id THEN
        RAISE EXCEPTION 'You cannot review your own account.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Create a trigger to call the function before inserting a new row into the rating table
CREATE TRIGGER prevent_self_review_trigger
BEFORE INSERT ON rating
FOR EACH ROW
EXECUTE FUNCTION prevent_self_review();

--TRIGGER13
CREATE OR REPLACE FUNCTION anonymize_user_data()
RETURNS TRIGGER AS $$
BEGIN

    UPDATE account
    SET username = 'deleted_user_' || NEW.id,
        email = 'deleted_user_' || NEW.id || '@example.com',
        password = 'deleted',
        profile_picture = NULL,
        birth_date = NULL,
        address = NULL
    WHERE id = OLD.id;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER anonymize_user_trigger
AFTER DELETE ON users
FOR EACH ROW
EXECUTE FUNCTION anonymize_user_data();

--TRIGGER14
CREATE OR REPLACE FUNCTION check_auction_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.end_date <= NEW.start_date + INTERVAL '1 day' THEN
        RAISE EXCEPTION 'The auction end date must be at least one day after the start date.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_auction_dates_trigger
BEFORE INSERT OR UPDATE ON auction
FOR EACH ROW
EXECUTE FUNCTION check_auction_dates();
```

A.2. Database population

```
SET search_path TO lbaw2451;

-- Populate accounts table
INSERT INTO account (username, email, password, registration_date, profile_picture, birth_date, address) VALUES
('john_doe', 'john.doe@example.com', 'hashed_password_1', '2024-01-10 08:30:00', 'profile1.jpg', '1990-05-15', '123 M
('jane_smith', 'jane.smith@example.com', 'hashed_password_2', '2024-01-11 09:00:00', 'profile2.jpg', '1995-08-20', '4
('admin_user', 'admin@example.com', 'hashed_password_admin', '2024-01-05 07:45:00', 'admin.jpg', '1988-11-30', '789 M

-- Populate users table
INSERT INTO users (id, is_deleted) VALUES
(1, FALSE), -- john_doe
(2, FALSE); -- jane_smith
```

Revision history

There have been no changes to the first submission.

GROUP2451, 03/10/2024

- Group member 1 Alexandre Silva, up202206633@up.pt (editor)
- Group member 2 Eduardo Baltazar, up202206313@up.pt
- Group member 3 Tiago Lourenço, up202004374@up.pt
- Group member 4 Tiago Aleixo, up202004996@up.pt