EBD: Database Specification Component

> Bidding the future and Selling the past.

A4: Conceptual Data Model

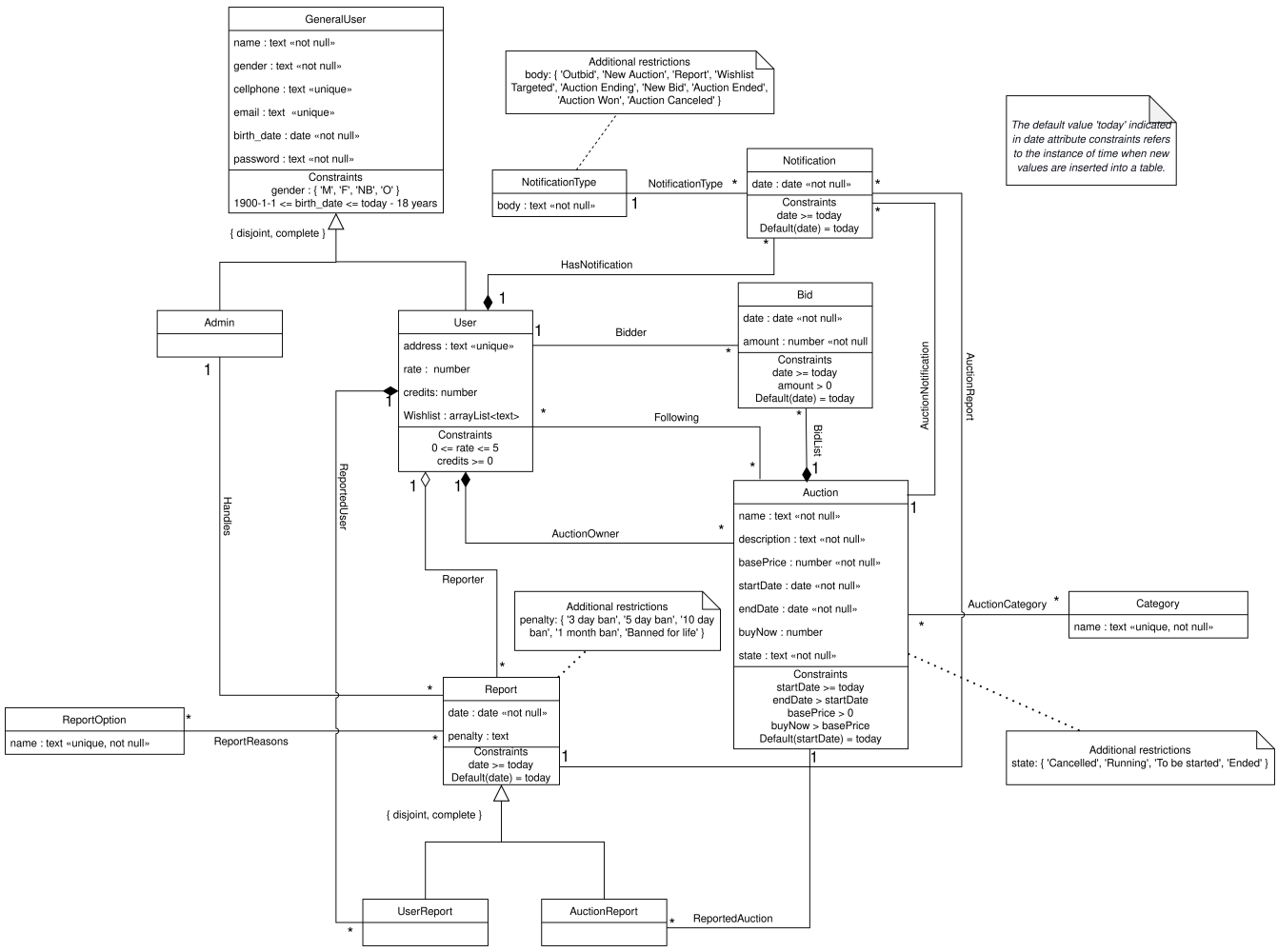
> The Conceptual Data Model contains the identification and description of the entities and relationships that are relevant to the database specification.

> A UML class diagram is used to document the model.

1. Class diagram

> UML class diagram containing the classes, associations, multiplicity and roles.
> For each class, the attributes, associations and constraints are included in the class diagram.

> Additional restrictions and atribute domains that didn't fit in the class box were written in uml notes.



2. Additional Business Rules

- BR01. Administrator accounts are independent of the user accounts, i.e. they cannot create or participate in auctions.
- BR02. An auction can only be cancelled by its owner if there are no bids.

- BR03. A user cannot bid on its own auction.
- BR04. When an account gets deleted, user activity is saved, but all personal data is erased.
- BR05. Items considered illegal to be sold (in the respective countries) can't be auctioned.
- BR06. The ending date of a given auction, as well as its biddings, is greater or equal then the starting date.
- BR07. Bids can only be done while the auction is active.
- BR08. A user can not delete an account if any of its bids are the highest in an active auction.
- BR09. Only valid users (not deleted accounts) can bid.
- BR10. Bidders can not make bids below the currently highest bid.

A5: Relational Schema, validation and schema refinement

> This artifact contains the Relational Schema obtained by mapping from the Conceptual Data Model.

1. Relational Schema

> The Relational Schema includes the relation schemas, attributes, domains, primary keys, foreign keys and other integrity rules: UNIQUE, DEFAULT, NOT NULL, CHECK.
> Relation schemas are specified in the compact notation:

Relation reference	**Relation Compact Notation**
R01	general_user(id, name **NN**, gender **CK** gender IN Gender, cellphone **UK**, email **UK**, birth_date **NN**, address **UK**, password **NN**, rate **CK** rate >= 0 AND rate <= 5, credits, wishlist, is_admin **NN**)
R02	bid(id, date **NN DF** Today, amount **NN**, user_id -> general_user **NN**, auction_id -> auction **NN**)
R03	notification (id, date **DF** Today **NN**, type **NN CK** type IN Notification_type, user_id -> general_user **NN**, auction_id -> auction, report_id -> report)
R04	auction (id, name **NN**, description **NN**, base_price **NN**, start_date **NN DF** Today, end_date **NN CK** start_date < end_date, buy_now, state **NN**, auction_owner_id -> general_user **NN**)
R05	category (id, name **NN UK**)
R06	auction_category (category_id -> category, auction_id -> auction)
R07	following (user_id -> general_user, auction_id-> auction)
R08	report (id, date **DF** Today **NN**, penalty **CK** penalty IN Penalty, reported_user -> general_user, reporter -> general_user **NN CK** reported_user != reporter, auction_reported -> auction, admin_id -> general_user)
R09	report_option (id, name **NN UK**)
R10	report_reasons (id_report_option -> report_option, id_report -> report)

- Legend:
  - **UK** = UNIQUE KEY

- **NN** = NOT NULL
- **DF** = DEFAULT
- **CK** = CHECK

Justification for Generalizations

Generalization	Justification
User / Admin / General User	Since the differences between an User and an Admin are few, we chose to generalize this using null fields in certain columns, namely: address, rate, credits and wishlist. Furthermore, we added a collumn named "isAdmin" that is composed of boolean values and is used to check if a given user is an Admin or not.
Reports / Auction Reports / User Reports	Once again Auction reports share the vast majority of their attributes and therefore it makes more sense to simply use nulls to express the difference between both of them.

2. Domains

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

Domain Name	Domain Specification
Today	DATE DEFAULT CURRENT_DATE
notification_type	ENUM ('Outbid', 'New Auction', 'Report', 'Wishlist Targeted', ‘Auction Ending’, ‘New Bid’, ‘Auction Ended’, ‘Auction Won’, ‘Auction Canceled’)
auction_possible_state	ENUM ('Cancelled', 'Running', 'To be started', 'Ended')
penalty_type	ENUM ('3 day ban', '5 day ban', '10 day ban', '1 month ban', 'Banned for life')
gender_possible	ENUM (‘M’, ‘F’, ‘NB’, ‘O’)

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. Should it be necessary, in case the scheme is not in the Boyce–Codd Normal Form (BCNF), the relational schema is refined using normalization.

Table R01 (general_user)	
Keys: {id}, {cellphone}, {email}, {address}	
Functional Dependencies	
FD0101	{id} -> {name, gender, cellphone, email, birth_date, address, password, rate, credits, wishlist, is_admin}
FD0102	{cellphone} -> {id, name, gender, email, birth_date, address, password, rate, credits, wishlist, is_admin}
FD0103	{email} -> {id, name, gender, cellphone, birth_date, address, password, rate, credits, wishlist, is_admin}
FD0104	{address} -> {id, name, gender, cellphone, email, birth_date, password, rate, credits, wishlist, is_admin}
Normal Form	BCNF

Table R02 (bid)	
Keys: {id}	
Functional Dependencies	
FD0201	{id} -> {date, amount, user_id -> general_user, auction_id -> auction}
Normal Form	BCNF

Table R03 (notification)	
Keys: {id}	
Functional Dependencies	
FD0301	{id} -> {date, type, user_id -> general_user, auction_id -> auction, report_id -> report}
Normal Form	BCNF

Table R04 (auction)	
Keys: {id}	
Functional Dependencies	
FD0401	{id} -> {name, description, base_price, start_date, end_date, buy_now, state, auction_owner_id -> general_user}
Normal Form	BCNF

Table R05 (category)	
Keys: {id}, {name}	
Functional Dependencies	
FD0501	{id} -> {name}
FD0502	{name} -> {id}
Normal Form	BCNF

Table R06 (auction_category)	
Keys: {category_id, auction_id}	
Functional Dependencies	
FD0601	none
Normal Form	BCNF

Table R07 (following)	
Keys: {user_id, auction_id}	
Functional Dependencies	
FD0701	none
Normal Form	BCNF

Table R08 (report)	
Keys: {id}	
Functional Dependencies	
FD0801	{id} -> {date, penalty, reported_user -> general_user, reporter -> general_user, auction_reported -> auction, admin_id-> general_user}
Normal Form	BCNF

Table R09 (report_option)	
Keys: {id, name}	
Functional Dependencies	
FD0901	{id} -> {name}
FD0902	{name} -> {id}
Normal Form	BCNF

Table R10 (report_reasons)	
Keys: {id_report_option, id_report}	
Functional Dependencies	
FD1001	none
Normal Form	BCNF

A6: Indexes, triggers, transactions and database population

> This artifact contains the physical schema of the database, the identification and characterisation of the indexes, the support of data integrity rules with triggers and the definition of the database user-defined functions.

> Furthermore, it also shows the database transactions needed to assure the integrity of the data in the presence of concurrent accesses. For each transaction, the isolation level is explicitly stated and justified.

> This artifact also contains the database's workload as well as the complete database creation script, including all SQL necessary to define all integrity constraints, indexes and triggers. Finally, this artifact also includes a separate script with INSERT statements to populate the database.

1. Database Workload

> A study of the predicted system load (database load). Estimate of tuples at each relation.

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	general_user	10 k	10 per day
R02	bid	100 k	100 per day
R03	notification	1 M	thousands per day
R04	auction	1 k	1 per day
R05	category	10	1 per month
R06	auction_category	1 k	1 per day

Relation reference	Relation Name	Order of magnitude	Estimated growth
R07	following	10 k	10 per day
R08	report	1 k	1 per week
R09	report_option	10	no growth
R10	report_reason	1 k	1 per week

2. Proposed Indexes

2.1. Performance Indexes

> Indexes proposed to improve performance of the identified queries.

Index	IDX01
Index relation	notifications
Index attribute	user_id
Index type	Hash
Cardinality	Medium
Clustering	No
Justification	Table 'notification' is very large, and it will be frequently queried to gather all the tuples associated with a certain user_id, therefore it is a good candidate to a hash type index, since the table will be queried using the equal operator (exact match). We also thought about applying clustering to this table, but since clustering is a one time operation and this table grows quickly, it wouldn't have much impact.
SQL code	

```
CREATE INDEX IF NOT EXISTS notification_user_id ON notification USING
hash(user_id);
```

Index	IDX02
Index relation	bid
Index attributes	auction_id, amount
Index type	B-tree
Cardinality	Medium
Clustering	No
Justification	We expect the 'bid' table to be very large, therefore we think it is a good idea to apply an index on it. The most frequent queries will be to find out all the bids related to a certain auction (using the auction_id attribute) and to find the largest bid of that set (find the tuple with the highest value in the 'amount' collumn, within a certain set of bids of an auction). For these reasons, we chose to use a B-tree index, since it will allow us to find the highest bid faster, while having a good time efficiency in finding the tuples with a given auction_id (logarithmic time).
SQL code	

```
CREATE INDEX IF NOT EXISTS bid_auction_id_amount ON bid USING
BTREE(auction_id, amount);
```

Index	IDX03
Index relation	general_user
Index attributes	wishlist
Index type	GIN
Cardinality	Medium
Clustering	No
Justification	Table 'general_user' will be regularly queried, especially on functionalities related to the user's wishlist. The 'wishlist' attribute is an array of strings, hence why we use a GIN type index, to efficiently handle queries that test for the presence of a specific string in the array.
SQL code	

```
CREATE INDEX IF NOT EXISTS user_wishlist ON general_user USING
GIN(wishlist);
```

2.2. Full-text Search Indexes

Index	IDX11
Index relation	auction
Index attributes	auction_tokens
Index type	GIN
Cardinality	Medium
Clustering	No
Justification	Since our Web Application will have a search engine to find auctions, having an index for full-text search on the auctions' name and description (the name having more weight than the description) was an obvious addition to increase the performance of user searches. Due to the auctions' static nature we are using the GIN index type which has better perfomance for this particular type of data.
SQL code	

```
-- Add new column in auction for tsvectors
ALTER TABLE auction
ADD COLUMN auction_tokens TSVECTOR;

-- Update tsvectors in auction table
UPDATE auction d1
SET auction_tokens = (setweight(to_tsvector('english', coalesce(d1.name,
'')), 'A') || setweight(to_tsvector('english', coalesce(d1.description,
'')), 'B'))
FROM auction d2;

-- Function to automatically update auction_tokens
CREATE OR REPLACE FUNCTION auction_tokens_update() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        NEW.auction_tokens = (setweight(to_tsvector('english',
coalesce(NEW.name, '')), 'A') || setweight(to_tsvector('english',
coalesce(NEW.description, '')), 'B'));
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.name <> OLD.title OR NEW.description <> OLD.description)
THEN
            NEW.auction_tokens = (setweight(to_tsvector('english',
coalesce(NEW.name, '')), 'A') || setweight(to_tsvector('english',
coalesce(NEW.description, '')), 'B'));
        END IF;
```

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

-- Create trigger before insert or update on auction
CREATE TRIGGER auction_tokens_update
    BEFORE INSERT OR UPDATE ON auction
    FOR EACH ROW
    EXECUTE PROCEDURE auction_tokens_update();

-- Create an index on the ts_vectors.
CREATE INDEX idx_auctions ON auction USING GIN(auction_tokens);
```

3. Triggers

> User-defined functions and trigger procedures that add control structures to the SQL language or perform complex computations, are identified and described to be trusted by the database server. Every kind of function (SQL functions, Stored procedures, Trigger procedures) can take base types, composite types, or combinations of these as arguments (parameters). In addition, every kind of function can return a base type or a composite type. Functions can also be defined to return sets of base or composite values.

> The justification behind all the triggers listed below is that they are needed in order to assure the database is consistent and its data is valid. Furthermore, they are also used to enforce some business rules (specified in the description of each trigger).

Trigger	TRIGGER01
Description	A user cannot bid on his own auction (BR03)
SQL code	

```
CREATE OR REPLACE FUNCTION bid_owner() RETURNS TRIGGER AS $BODY$ BEGIN IF
EXISTS (
                SELECT *
                FROM auction
                WHERE NEW.auction_id = id
                        AND NEW.user_id = auction_owner_id
        ) THEN RAISE EXCEPTION 'A user cannot bid on his own auction.';
END IF;
RETURN NEW;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS bid_owner ON bid;
CREATE TRIGGER bid_owner BEFORE
INSERT ON bid FOR EACH ROW EXECUTE PROCEDURE bid_owner();
```

Trigger	TRIGGER02
Description	Admins cannot bid. (BR01)
SQL code	

```
CREATE OR REPLACE FUNCTION bid_admin() RETURNS TRIGGER AS $BODY$ BEGIN IF
EXISTS (
                SELECT *
                FROM general_user
                WHERE NEW.user_id = id
                        AND is_admin = TRUE
        ) THEN RAISE EXCEPTION 'An Admin cannot bid.';
END IF;
RETURN NEW;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS bid_admin ON bid;
CREATE TRIGGER bid_admin BEFORE
INSERT ON bid FOR EACH ROW EXECUTE PROCEDURE bid_admin();
```

Trigger	TRIGGER03
Description	Bids can only be done while the auction is active. (BR06)
SQL code	

```
CREATE OR REPLACE FUNCTION bid_date() RETURNS TRIGGER AS $BODY$ BEGIN IF
EXISTS (
                SELECT *
                FROM auction
                WHERE NEW.auction_id = id
                        AND (
                                NEW.date > end_date
                                OR NEW.date < start_date
                        )
        ) THEN RAISE EXCEPTION 'Invalid Date.';
END IF;
RETURN NEW;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS bid_date ON bid;
CREATE TRIGGER bid_date BEFORE
INSERT ON bid FOR EACH ROW EXECUTE PROCEDURE bid_date();
```

Trigger	TRIGGER04 and TRIGGER05
Description	A user can not delete an account if any of its bids are the highest in an active auction. When a user account is deleted, all the personal information is erased but its activity remains in the system.
SQL code	

```
CREATE OR REPLACE FUNCTION stop_delete_users() RETURNS TRIGGER AS $BODY$
BEGIN IF EXISTS (
                SELECT *
                FROM auction,
                        bid AS current_bid
                WHERE bid.auction_id == auction.id
                        AND auction.state == 'Running'
                        AND NOT EXISTS (
                                SELECT bid.amount
                                FROM bid
                                where bid.amount > current_bid.amount
                        )
                        AND current_bid.user_id == OLD.user_id
        ) THEN RAISE EXCEPTION 'You can not delete your account while you
have the highest bidding in an active auction.';
END IF;
UPDATE bid
SET name = "Deleted Account",
        email = NULL,
        gender = NULL,
        cellphone = NULL,
        birth_date = NULL,
        address = NULL,
        rate = NULL,
        credits = NULL,
        wishlist = NULL
WHERE id == OLD.id;
RETURN NULL;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS delete_users ON general_user;
CREATE TRIGGER delete_users BEFORE DELETE ON general_user EXECUTE PROCEDURE
stop_delete_users();
```

Trigger	TRIGGER06
Description	Bidders can not make bids below the currently highest bid.
SQL code	

```
CREATE OR REPLACE FUNCTION check_max_bid() RETURNS TRIGGER AS $BODY$ BEGIN
IF EXISTS (
                SELECT *
                FROM bid
                WHERE bid.auction_id = NEW.auction_id
                        AND bid.amount >= NEW.amount
        ) THEN RAISE EXCEPTION 'Bid is lower than the highest bid.';
END IF;
RETURN NEW;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS bid_lower_than_max ON bid;
CREATE TRIGGER bid_lower_than_max BEFORE
INSERT ON bid FOR EACH ROW EXECUTE PROCEDURE check_max_bid();
```

Trigger	TRIGGER07
Description	Only valid users (not deleted accounts) can bid.
SQL code	

```
CREATE OR REPLACE FUNCTION check_bid_user_exists() RETURNS TRIGGER AS
$BODY$ BEGIN IF NOT EXISTS (
                SELECT *
                FROM general_user
                WHERE id == NEW.id AND email IS NOT NULL
        ) THEN RAISE EXCEPTION 'User not found.';
END IF;
RETURN NEW;
END $BODY$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS check_bid_user_exists ON bid;
CREATE TRIGGER check_bid_user_exists BEFORE
INSERT ON bid FOR EACH ROW EXECUTE PROCEDURE check_bid_user_exists();
```

4. Transactions

> Transactions needed to assure the integrity of the data.

Transaction	TRAN01
Description	Create a new auction with category
Justification	When creating an auction, the auction owner must input the auction category. If, for some reason, in the time between retrieving all possible auction categories and submitting the auction to the database, the auction category is eliminated by an admin or something happens to the server then the auction would be created without an auction_category which would eventually break the database and lead to phantom reads.
SQL code	

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT * FROM auction_category;

-- Insert Auction
INSERT INTO auction (id, name, description, base_price, start_date,
end_date, buy_now, state, auction_owner_id )
VALUES ($name, $description, $base_price, $start_date, $end_date, $buy_now,
$state, $auction_owner_id);

-- Insert Auction Category
INSERT INTO auction_category (category_id, auction_id)
VALUES ($category_id, $auction_id);


END TRANSACTION;
```

Transaction	TRAN02
Description	Auction end, credit transition
Justification	When an auction ends, the auction owner should be awarded the value of the highest bid and the highest bidder should see the amount of the bid removed from the credits. A interruption in the program could lead to excess credits in one or the other resulting in an unstable intermediate state in the database.
SQL code	

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

UPDATE auction SET state = "Ended"
    WHERE id = $auction_id;

-- Add funds to auction owner
UPDATE user SET credits = credits + (SELECT value from bid WHERE id =
$bid_id)
    WHERE id = $auction_owner_id;

-- Remove funds from winning bidder
UPDATE user SET credits = credits - (SELECT value from bid WHERE id =
$bid_id)
    WHERE user = (SELECT user_id from bid WHERE id = $bid_id);

END TRANSACTION;
```

Transaction	TRAN03
Description	Auction start, notification issued
Justification	When creating an auction there should be issued a notification to all the users that have the title of the auction in their wishlist. If after selecting the user a competing operation changes the users in the database or their wishlist, we can have a phantom read. Therefore we should perform this action as a block also protecting us from creating an auction and due to some failure the part where the notification is issued not being performed.
SQL code	

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-- Auction Started
UPDATE auction SET state = "Running"
    WHERE id = $auction_id;

-- Find the Users that should be receiving the notification
SELECT id FROM users WHERE $auction_title = ANY(wishlist);

-- Issue the notification
INSERT INTO notification (id, date, type, user_id, auction_id, report_id)
VALUES ($date, $type, $user_id, $auction_id, NULL);

END TRANSACTION;
```

Annex A. SQL Code

> The database scripts are included in this annex to the EBD component.
>
> The database creation script and the population script should be presented as separate elements. The creation script includes the code necessary to build (and rebuild) the database. The population script includes an amount of tuples suitable for testing and with plausible values for the fields of the database.
>
> The complete code of each script must be included in the group's git repository and links added here.

A.1. Database schema

> The complete database creation must be included here and also as a script in the repository.

```
SET search_path TO lbaw2271;

DROP TABLE IF EXISTS auction_category;
DROP TABLE IF EXISTS bid;
DROP TABLE IF EXISTS category;
DROP TABLE IF EXISTS following;
DROP TABLE IF EXISTS notification;
DROP TABLE IF EXISTS report_reasons;
DROP TABLE IF EXISTS report;
DROP TABLE IF EXISTS report_option;
DROP TABLE IF EXISTS auction;
DROP TABLE IF EXISTS general_user;

DROP TYPE IF EXISTS notification_type;
DROP TYPE IF EXISTS penalty_type;
DROP TYPE IF EXISTS auction_possible_state;
DROP TYPE IF EXISTS gender_possible;


CREATE TYPE notification_type AS ENUM ('Outbid', 'New Auction', 'Report',
'Wishlist Targeted', 'Auction Ending', 'New Bid', 'Auction Ended', 'Auction
Won', 'Auction Canceled');
CREATE TYPE auction_possible_state AS ENUM ('Cancelled', 'Running', 'To be
started', 'Ended');
CREATE TYPE penalty_type AS ENUM ('3 day ban', '5 day ban', '10 day ban',
'1 month ban', 'Banned for life');
CREATE TYPE gender_possible AS ENUM ('M', 'F', 'NB', 'O');

CREATE TABLE IF NOT EXISTS general_user (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    gender gender_possible,
    cellphone CHAR(9) UNIQUE,
    email VARCHAR(320) UNIQUE,
    birth_date DATE NOT NULL,
    address VARCHAR(255) UNIQUE,
```

```
    password VARCHAR NOT NULL,
    rate REAL,
    credits REAL,
    wishlist TEXT [],
    is_admin BOOLEAN NOT NULL,
    CONSTRAINT valid_rate CHECK (rate >= 0 AND rate <= 5),
    CONSTRAINT valid_birth CHECK (birth_date between '1900-01-01' and now()
- interval '18 years')
);

CREATE TABLE IF NOT EXISTS auction (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    base_price REAL NOT NULL,
    start_date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    end_date TIMESTAMP WITH TIME ZONE NOT NULL,
    buy_now REAL,
    state auction_possible_state NOT NULL,
    auction_owner_id INTEGER REFERENCES general_user ON UPDATE CASCADE NOT
NULL,
    CONSTRAINT valid_dates CHECK (start_date < end_date)
);

CREATE TABLE IF NOT EXISTS bid (
    id SERIAL PRIMARY KEY,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    amount REAL NOT NULL,
    user_id INTEGER REFERENCES general_user ON UPDATE CASCADE NOT NULL,
    auction_id INTEGER REFERENCES auction ON UPDATE CASCADE ON DELETE
CASCADE NOT NULL
);

CREATE TABLE IF NOT EXISTS report (
    id SERIAL PRIMARY KEY,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    penalty penalty_type,
    reported_user INTEGER REFERENCES general_user ON UPDATE CASCADE ON
DELETE CASCADE,
    reporter INTEGER REFERENCES general_user ON UPDATE CASCADE NOT NULL,
    auction_reported INTEGER REFERENCES auction ON UPDATE CASCADE,
    admin_id INTEGER REFERENCES general_user ON UPDATE CASCADE,
    CONSTRAINT no_self_reports CHECK (reported_user != reporter)
);

CREATE TABLE IF NOT EXISTS notification (
    id SERIAL PRIMARY KEY,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    TYPE notification_type NOT NULL,
    user_id INTEGER REFERENCES general_user ON UPDATE CASCADE ON DELETE
CASCADE NOT NULL,
    auction_id INTEGER REFERENCES auction ON UPDATE CASCADE,
    report_id INTEGER REFERENCES report ON UPDATE CASCADE
);
```

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

CREATE TABLE IF NOT EXISTS auction_category (
    category_id INTEGER REFERENCES category ON UPDATE CASCADE,
    auction_id INTEGER REFERENCES auction ON UPDATE CASCADE,
    PRIMARY KEY (category_id, auction_id)
);

CREATE TABLE IF NOT EXISTS following (
    user_id INTEGER REFERENCES general_user ON UPDATE CASCADE,
    auction_id INTEGER REFERENCES auction ON UPDATE CASCADE,
    PRIMARY KEY (user_id, auction_id)
);

CREATE TABLE IF NOT EXISTS report_option (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS report_reasons (
    id_report_option INTEGER REFERENCES report_option ON UPDATE CASCADE,
    id_report INTEGER REFERENCES report ON UPDATE CASCADE,
    PRIMARY KEY (id_report_option, id_report)
);
```

A.2. Database population

```
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(1, 'Rafael
Pinta', 'M', '913547483', 'rafaelpinta@fe.up.pt', '1982-05-14', '271
Unnamed 051',
'$2b$12$CIaa06HtI/stVPNQp3hb2OXSmBvDUkPzzGQP8rvLpBaVQdr6QhayW', 3.6, 2212,
ARRAY []::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(2, 'Quintin
Fankhanel', 'M', '929138897', 'quintinfankhanel@fe.up.pt', '1964-04-06',
'388 Wayland st',
'$2b$12$IGg02jV1OffqddVLwDG4Ou8JlDRrDiPDl8jxEhe1gYYtBAF4N90dS', 5, 3016,
ARRAY ['records']::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(3, 'Earnestine
Morado', 'F', '965502523', 'earnestine.morado@gmail.com', '1967-12-28',
'168 Weston ct',
'$2b$12$N5M.PEHCHcKwBTy5UeGyg.FtcSJ3K2HfRKaklRFYuXjolE4wQX9HG', 5, 1589,
ARRAY ['dolls', 'baseball cards', 'trading cards']::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(4, 'Selina
```

```
Jankoviak', 'F', '931886980', 'selinajankoviak@icloud.com', '1979-12-19',
'671 Peacemakers st',
'$2b$12$V3hWJJxEGAjGPrGJWlk/t.jvfGYWij7xlNqzaz967BYtXG6pV1bYO', 5, 2045,
ARRAY ['records', 'coin', 'diamond', 'vinyl', 'currency']::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(5, 'Auther
Warnell', 'M', '913664514', 'auther.warnell@icloud.com', '1996-06-28', '388
Le conte ave',
'$2b$12$fz9jEFspK2zbbDYgpji/w.ZpzIlveKNW39vADLlxDndyhH/YUo.Xa', 3.0, 597,
ARRAY []::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(6, 'Geraldine
Siggers', 'F', '93635200', 'geraldinesiggers@yahoo.com', '1971-09-16', '747
Skyline blvd',
'$2b$12$H7SsXwk/t2PmPnGS8w6uhuigxL56i7BoF.ZoOsQMLUI28JwF0vk9G', 4.5, 1713,
ARRAY []::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(7, 'Reno
Pleites', 'M', '961273494', 'renopleites@icloud.com', '1964-07-01', '178
Gould st', '$2b$12$LEvDsol8hbuu6CtdjfPbSuqeW75t9JXUOhTn.ATpa1MkmPdKT7GBC',
4.1, 2928, ARRAY []::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(8, 'Annie
Mccally', 'F', '962094294', 'anniemccally@fe.up.pt', '1977-06-08', '64
Ortega way',
'$2b$12$EvuSdCHCgH2CBloMPnjLNOnRFujn4JnIBALfaRjVHig91CVMi.NZ.', 3.1, 2782,
ARRAY ['nb550', 'diamond', 'trading cards', 'rolex']::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(9, 'Quentin
Kola', 'M', '932400131', 'quentinkola@yahoo.com', '1972-04-05', '21 Merced
ave', '$2b$12$55PeOaSFlZ8Lbq3SlYFozepraMiCVCHBGAp.6sK60TQSlPxvRI5hy', 4.6,
2177, ARRAY []::text[], FALSE);
insert into general_user(id, name, gender, cellphone, email, birth_date,
address, password, rate, credits, wishlist, is_admin) values(10, 'Hal
Alcius', 'M', '936711449', 'halalcius@fe.up.pt', '1991-02-19', '415
Cornwall st',
'$2b$12$eOKxR3z0n6QdPkgpZf8djOtl8zp0LltlS3K9Kt8xZXDJ4dNiw0dFW', 2.1, 1816,
ARRAY ['comic', 'dolls', 'art', 'doll', 'stamps']::text[], FALSE);
```

Revision history

Changes made to the first submission:

1. Item 1
2. ..

GROUP2271, 30/10/2022

- André Sousa, up202005277@fe.up.pt (Editor)
- Pedro Moreira, up201905429@fe.up.pt

- Pedro Fonseca, up202008307@fe.up.pt
- Vítor Cavaleiro, up202004724@edu.fe.up.pt