A6: Indexes, triggers, user functions and population

This artefact 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. This artefact 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.

1. Database Workload

> A study of the predicted system load (database load), organized in subsections.

1.1. Tuple Estimation

> Estimate of tuples at each relation.

Relation reference	**Relation Name**	**Order of magnitude**	**Estimated growth**
R01	Auction	tens of thousands	hundreds per week
R02	User	thousands	dozens per day
R03	Add_Credits	tens of thousands	hundreds per day
R04	Bid	hundreds of thousands	thousands per day
R05	Category	units	no growth
R06	City	thousands	units per month
R07	Comment	hundreds	dozens per day

R08	Country	hundreds	no growth
R09	Notification	hundreds of thousands	thousands per day
R10	Report	hundreds	dozens per month
R11	CategoryOfAuction	units	units per day
R12	Blocks	dozens	units per month
R13	Edit_Moderator	dozens	units per month
R14	Edit_Categories	units	no growth

1.2. Frequent Queries

> Most important queries (SELECT) and their frequency.

Query reference	SELECT01
Query description	User's login
Query frequency	hundreds per day

```
SELECT password
FROM "user"
WHERE "user".username = $username;


CREATE TEMPORARY TABLE current_user(username varchar);
INSERT INTO current_user(username) VALUES ($username);
```

Query reference	SELECT02
Query description	User's profile information
Query frequency	hundreds per day

```
  SELECT username, pathToPhoto, email, "/rating", balance,
phonenumber
  FROM "user"
  WHERE "user".ID = $userid;


  SELECT "auction".ID, "auction".pathToPhoto, title,
auctionCreator, "bid".value, "bid".date
  FROM "auction", "bid", "user"
  WHERE "user".ID = $userid AND "auction".ID =
"bid".auctionbidded AND "user".ID = "bid".bidder;


  SELECT "auction".ID, "auction".pathToPhoto, title,
auctionCreator, "bid".value, "bid".date
  FROM "auction", "bid", "user"
  WHERE "user".ID = $userid AND "auction".ID =
"bid".auctionbidded AND "user".ID = "bid".bidder;
```

Query reference	SELECT03
Query description	See an auction
Query frequency	hundreds per day

```
  SELECT title, description, sellingReason, pathToPhoto,
startingPrice, minimumSellingPrice, buyNow, startDate,
limitDate, auctionCreator
  FROM "auction"
  WHERE "auction".ID = $auctionid;


  SELECT MAX(value) as CurrentBid
  FROM "bid"
  WHERE "bid".auctionBidded=$auctionid;


  SELECT username, pathToPhoto, email, "/rating", balance
  FROM "user"
  WHERE "user".ID = auctionCreator;
```

Query reference	SELECT04
Query description	Get auctions where user made a bid.
Query frequency	hundreds per day

```
  SELECT "auction".ID, "auction".pathToPhoto, title,
auctionCreator, "bid".value, "bid".date
  FROM "auction", "bid", "user"
  WHERE "user".ID = $userid AND "auction".ID =
"bid".auctionbidded AND "user".ID = "bid".bidder;


  SELECT MAX(value) as CurrentBid
```

```
FROM "bid"
WHERE "bid".auctionBidded=$auctionid;
```

Query reference	SELECT05
Query description	Get auctions associated to category.
Query frequency	hundreds per day

```
SELECT title, pathToPhoto, startDate, limitDate, numberOfBids
FROM "auction", "categoryofauction"
WHERE "auction".ID = "categoryofauction".auction AND
"categoryofauction".category = $categoryid AND
"auction".state='Active' LIMIT ALL OFFSET 0;
```

Query reference	SELECT06
Query description	Get auction's highest bidder.
Query frequency	hundreds per week

```
SELECT bidder
FROM "bid"
WHERE "bid".auctionBidded=$auctionid
ORDER BY value DESC LIMIT 1;
```

Query reference	SELECT07
Query description	Get user's auctions he won
Query frequency	hundreds per month

```
SELECT pathToPhoto, title, finalPrice, finalDate
FROM "auction"
WHERE  "auction".auctionWinner = $userId;
```

Query reference	SELECT08
Query description	Get user's "sold" auctions
Query frequency	hundreds per month

```
SELECT pathToPhoto, title, finalPrice, finalDate
FROM "auction"
WHERE  "auction".auctionCreator = $userId AND
"auction".state='Over'::auctionstate;
```

Query reference	SELECT9
Query description	Get moderator's active auctions
Query frequency	hundreds day

```
  SELECT ID, pathToPhoto, title, auctionCreator,
minimumSellingPrice, buyNow, limitDate
  FROM "auction"
  WHERE responsibleModerator  = $ID AND
state='Active'::auctionstate;


  SELECT MAX(value) as CurrentBid
  FROM "bid"
  WHERE "bid".auctionBidded = $auctionId;
```

Query reference	SELECT10
Query description	Get pending auctions
Query frequency	hundreds per day

```
  SELECT ID, pathToPhoto, title, auctionCreator,
minimumSellingPrice, buyNow, limitDate
  FROM "auction"
  WHERE state='Pending'::auctionstate;
```

Query reference	SELECT11
Query description	Get user's notifications
Query frequency	dozens per day

```
  SELECT type
  FROM "notification"
  WHERE "notification".authenticatedUserID  = $ID;
```

Query reference	SELECT12
Query description	Search an auction/user based on some words
Query frequency	hundreds per day

```
  SELECT ID, title, description
  From "auction"
  WHERE (title ILIKE '%$search%' or description ILIKE
'%$search%')and state = 'Active'


  SELECT ID, username
  From "user"
  WHERE username ILIKE '%$search%' or completename ILIKE
'%$search%'
```

1.3. Frequent Updates

> Most important updates (INSERT, UPDATE, DELETE) and their frequency.

Query reference	UPDATE01
Query description	Make Auction active
Query frequency	hundreds per week

```
  UPDATE "auction"
  SET state = 'Active', responsibleModerator =
$responsibleModerator
  WHERE ID = $ID;
```

Query reference	UPDATE02
Query description	Finalize Auction
Query frequency	hundreds per week

```
  UPDATE "auction"
  SET state='Over', auctionWinner = $winner, finalPrice =
$finalPrice, finalDate= $finalDate
  WHERE ID = $ID;
```

Query reference	UPDATE03
Query description	Rate Auction
Query frequency	hundreds per week

```
UPDATE "auction"
SET rate = $rate
WHERE ID = $ID;
```

Query reference	UPDATE04
Query description	Update Credit
Query frequency	dozens per month

```
UPDATE "user"
SET balance = $credit
WHERE ID = $ID;
```

Query reference	UPDATE05
Query description	Unblock/Block a previous blocked user
Query frequency	units per month

```
UPDATE "blocks"
SET state = $state
WHERE blocked  = $ID;
```

Query reference	UPDATE06
Query description	Update user's profile photo
Query frequency	units per month

```
UPDATE "user"
SET pathToPhoto = $pathToPhoto
WHERE ID = $ID;
```

Query reference	UPDATE07
Query description	Update user's location

Query frequency	units per month

```
  UPDATE "user"
  SET address = $address, postalCode = $postalCode, city =
$city
  WHERE ID = $ID;
```

Query reference	UPDATE08
Query description	Update user's password
Query frequency	units per month

```
  UPDATE "user"
  SET password = $password
  WHERE ID = $ID;
```

Query reference	UPDATE09
Query description	Mark a notification as read.
Query frequency	dozens per day

```
UPDATE "notification"
SET read = true
WHERE ID = $ID;
```

Query reference	INSERT01
Query description	New user registered
Query frequency	dozens per day

```
  INSERT INTO "user" (typeOfUser, username, password,
pathToPhoto, completeName, email, birthDate, "/rating",
address, postalCode, balance, city, phonenumber, blocked)
  VALUES ('Normal', $username, $password, $pathToPhoto,
$completeName, $email, $birthDate, NULL, $address, $postalCode,
0, $city, $phonenumber, false);
```

Query reference	INSERT02
Query description	New moderator registered
Query frequency	units per month

```
  INSERT INTO "user" (typeOfUser, username, password,
pathToPhoto, completeName, email, birthDate, "/rating",
address, postalCode, balance, city)
  VALUES ('Moderator', $username, $password, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, false);
```

Query reference	INSERT03
Query description	Create an Auction
Query frequency	hundreds per week

```
  INSERT INTO "auction" (state, title, description,
sellingReason, pathToPhoto, startingPrice, minimumSellingPrice,
buyNow, startDate, limitDate, refusalDate, "/numberOfBids",
reasonOfRefusal, auctionWinner, auctionCreator,
responsibleModerator, rate, finalPrice, finalDate)
  VALUES ('Pending', $title, $description, $sellingReason,
$pathToPhoto, $startingPrice, $minimumSellingPrice, $buyNow,
$startDate, $limitDate, NULL, 0, NULL, NULL, $auctionCreator,
NULL, NULL, NULL, NULL);
```

Query reference	INSERT04

Query description	Bid
Query frequency	hundreds per day

```
  INSERT INTO "bid" (auctionBidded, bidder, date, value,
isBuyNow)
  VALUES ($auctionBidded, $bidder, $date, $value, $isBuyNow);
```

Query reference	INSERT05
Query description	Add Credit
Query frequency	dozens per month

```
  INSERT INTO "add_Credits" (value, date, user, paypalID)
  VALUES ($value, $date, $user, $paypalID);
```

Query reference	INSERT06
Query description	Add Category
Query frequency	units per year

```
INSERT INTO "category" (name, parent)
VALUES ($name, $parent);
```

Query reference	INSERT07
Query description	Make a report
Query frequency	dozens per month

```
INSERT INTO "report" (auctionID, normalUserID, date, reason)
VALUES ($auctionID, $normalUserID, $date, $reason);
```

Query reference	INSERT08
Query description	Comment an auction
Query frequency	dozens per day

```
INSERT INTO "comment" (date, description, auctionCommented,
userCommenter)
VALUES ($date, $description, $auctionCommented,
$userCommenter);
```

Query reference	INSERT09
Query description	Block an user
Query frequency	units per month

```
  INSERT INTO "blocks" (blocked, blocker, state, description,
date)
  VALUES ($blocked, $blocker, 'Blocked', $description, $date);
```

Query reference	DELETE01
Query description	Delete a comment
Query frequency	units per month

```
  DELETE FROM "comment"
  WHERE ID = $ID;
```

2. Proposed Indexes

2.1. Performance Indexes

> Indexes proposed to improve performance of the identified queries.

Index reference	IDX01
Related queries	SELECT02
Index relation	User
Index attribute	id
Index type	Hash
Cardinality	high
Clustering	No
Justification	Used to search for the user's information. This query has to be fast because it's executed several times and the table has many entries. Cardinality is high due to the fact that id is unique and so it doesn't need ranged query support. This being said, clustering won't be used.

```
CREATE INDEX user_ind ON User USING hash (id);
```

Index reference	IDX02

Related queries	SELECT07
Index relation	bid
Index attribute	auctionbidded
Index type	B-tree
Cardinality	medium
Clustering	Yes
Justification	Used to search the bids of an auction and has to be fast because it's executed several times. Cardinality is medium so it's a good candidate for clustering.

```
CREATE INDEX bid_ind ON bid USING btree (auctionbidded);
```

Index reference	IDX03
Related queries	SELECT11
Index relation	auction
Index attribute	state
Index type	B-tree

Cardinality	low
Clustering	Yes
Justification	Used to search the pending auctions. Cardinality is low and it needs a ranged query so we use clustering.

```
CREATE INDEX pending_ind ON auction USING btree (state);
```

2.2. Full-text Search Indices

> The system being developed must provide full-text search features supported by PostgreSQL. Thus, it is necessary to specify the fields where full-text search will be available and the associated setup, namely all necessary configurations, indexes definitions and other relevant details.

Index reference	IDX04
Related queries	SELECT13
Index relation	auction
Index attribute	title
Index type	GiST
Clustering	No
	To improve the performance of full text searches while searching

Justification	for the titles of the auctions; GiST because it's better for dynamic data.

```
CREATE INDEX search_title_idx ON auction USING GIST
(to_tsvector('english', title)
```

Index reference	IDX05
Related queries	SELECT13
Index relation	User
Index attribute	username
Index type	GiST
Clustering	No
Justification	To improve the performance of full text searches while searching for the usernames of users; GiST because it's better for dynamic data.

```
CREATE INDEX search_username_idx ON User USING GIST
(to_tsvector('english', username)
```

Index reference	IDX06
Related queries	SELECT13
Index relation	User
Index attribute	completename
Index type	GiST
Clustering	No
Justification	To improve the performance of full text searches while searching for the names of users; GiST because it's better for dynamic data.

```
CREATE INDEX search_name_idx ON User USING GIST
(to_tsvector('english', completename)
```

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.

3.1 User-defined functions

UDF reference	UDF01
UDF description	Check if auction creator and auction winner (in case of existance) are Normal users and responsiblemoderator is Administrator or Moderator (in case of existance).

```
CREATE FUNCTION check_auction_users(auctioncreator integer,
auctionwinner integer, responsiblemoderator integer) RETURNS
boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE createrUser TypeOfUser;
DECLARE winnerUser TypeOfUser;
DECLARE responsibleUser TypeOfUser;
BEGIN
        SELECT typeOfUser INTO createrUser FROM User where id =
auctioncreator ;
        IF(auctionwinner IS NULL)
           THEN winnerUser := NULL ;
           ELSE SELECT typeOfUser INTO winnerUser FROM User
where id = auctionwinner ;
        END IF;
        IF(responsiblemoderator IS NULL)
           THEN responsibleUser := NULL ;
           ELSE SELECT typeOfUser INTO responsibleUser FROM
User where id = responsiblemoderator ;
        END IF;
        IF createrUser = 'Normal'::TypeOfUser AND (winnerUser =
'Normal'::TypeOfUser OR winnerUser IS NULL) AND (
responsibleUser = 'Moderator'::TypeOfUser OR responsibleUser =
'Administrator'::TypeOfUser OR responsibleUser IS NULL
)
```

```
                THEN passed := true;
                ELSE passed := false;
                END IF;
                RETURN passed;
END;$$;
```

UDF reference	UDF02
UDF description	Check in block association if user blocked is a Normal user and blocker is an Administrator or Moderator.

```
CREATE FUNCTION check_block_users(blocked integer, blocker
integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE blokedUser TypeOfUser;
DECLARE blokerUser TypeOfUser;
BEGIN
                SELECT typeOfUser INTO blokedUser FROM User
where id = blocked ;
                SELECT typeOfUser INTO blokerUser FROM User
where id = blocker ;
                IF blokedUser = 'Normal'::TypeOfUser AND
(blokerUser = 'Administrator'::TypeOfUser OR blokerUser =
'Moderator'::TypeOfUser)
                THEN passed := true;
                ELSE passed := false;
                END IF;
RETURN passed;
END;$$;
```

UDF reference	UDF03
UDF description	Check in edit moderators association if edited Moderator is a Moderator user and editor is an Administrator.

```
CREATE FUNCTION check_edit_moderators(removedmod integer,
removeradmin integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed boolean;
DECLARE removerModUser TypeOfUser;
DECLARE removerAdminUser TypeOfUser;
BEGIN
        SELECT typeOfUser INTO removerModUser FROM User where
id = removedMod ;
        SELECT typeOfUser INTO removerAdminUser FROM User where
id = removerAdmin ;
        IF removerModUser = 'Moderator'::TypeOfUser AND
removerAdminUser = 'Administrator'::TypeOfUser
        THEN passed := true;
        ELSE passed := false;
        END IF;
RETURN passed;
END;$$;
```

UDF reference	UDF04

UDF description	Check if rejected auction has date of rejection and reason.

```
CREATE FUNCTION check_rejected_auction(state auctionstate,
dateofrefusal timestamp with time zone, reasonofrefusal
character varying) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE pass boolean;
BEGIN
  IF state = 'Rejected'::auctionstate
      THEN IF dateOfRefusal IS NOT NULL AND reasonOfRefusal IS
NOT NULL
           THEN pass := true;
           ELSE pass := false;
           END IF;
  ELSE pass := true;
  END IF;
  RETURN pass;
END;$$;
```

UDF reference	UDF05
UDF description	Check in edit category association if editor is an Administrator.

```
CREATE FUNCTION check_admin_modify_category(admin integer)
RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE adminUser TypeOfUser;
```

```
BEGIN
SELECT typeOfUser INTO adminUser FROM User where id = admin ;
IF
adminUser = 'Administrator'::TypeOfUser THEN passed := true;
ELSE passed := false;
END IF;
RETURN passed;
END;
$$;
```

UDF reference	UDF06
UDF description	Check if an auction that is over has all obligatory camps not null.

```
CREATE FUNCTION check_auction_win(state auctionstate, finaldate
timestamp with time zone, finalprice integer, auctionwinner
integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE pass boolean;
BEGIN
  IF state = 'Over'::auctionstate
      THEN IF finaldate IS NOT NULL AND finalprice IS NOT NULL
AND auctionwinner IS NOT NULL
           THEN pass := true;
           ELSE pass := false;
           END IF;
  ELSE pass := true;
  END IF;
  RETURN pass;
END;$$;
```

UDF reference	UDF07
UDF description	Get the current user of the website for other functions checking.

```
CREATE FUNCTION get_current_user() RETURNS varchar
    LANGUAGE plpgsql
    AS $$DECLARE
    cur_user varchar;
BEGIN
    BEGIN
        cur_user := (SELECT "username" FROM curr_user);
    EXCEPTION WHEN undefined_table THEN
        cur_user := 'unknown_user';
    END;
    RETURN cur_user;
END;
$$;
```

3.2 Triggers

Trigger reference	TRIGGER01
Trigger description	The creator of an auction cannot make a bid on it!

```
CREATE FUNCTION auction_creator() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
  IF EXISTS (SELECT * FROM auction
    INNER JOIN "user" ON "user".id = auction.auctioncreator
    WHERE NEW.auctionbidded = auction.id AND NEW.bidder =
"user".id) THEN
    RAISE EXCEPTION 'The creator of an auction cannot make a
bid on it!';
  END IF;
  RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER auction_creator
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE PROCEDURE auction_creator();
```

Trigger reference	TRIGGER02
Trigger description	A bid on this auction has to have a greater value than a previous one on this auction. When this happens it updates the user that made the last bid with highest value so that its balance is restored and generates a notification for that user.

```
CREATE FUNCTION bid_greater_than_last() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
```

```
        IF EXISTS (SELECT * FROM bid WHERE NEW.auctionbidded =
auctionbidded AND NEW.value <= value)="" then="" raise=""
exception="" 'a="" bid="" on="" this="" auction="" has="" to=""
have="" a="" greater="" value="" than="" previous="" one=""
auction.';="" else="" update="" "user"="" set="" balance="
((SELECT" from="" "bid"="" where=""
"bid".auctionbidded="NEW.auctionbidded" order="" by="" desc=""
limit="" 1)="" +="" (select="" id="(SELECT" bidder=""
auctionbidded="NEW.auctionbidded" 1)))="" 1);="" insert=""
into="" notification="" (id,="" date,="" description,=""
type,="" auctionassociated,="" authenticated_userid)=""
values="" (default,="" transaction_timestamp(),="" 'your=""
was="" surpassed.="" try="" again!',="" 'bid="" exceeded',=""
new.auctionbidded,="" 1));="" end="" if;="" return="" new;=""
end$$;=""
-- installed as --
CREATE TRIGGER bid_greater_than_last
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE PROCEDURE bid_greater_than_last();
```

Trigger reference	TRIGGER03
Trigger description	Checks if bidder has enough balance to make a bid of this value and if it has discounts it on the balance

```
CREATE FUNCTION bidder_has_money() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
        IF EXISTS (SELECT * FROM User WHERE NEW.bidder = id AND
NEW.value > balance)          THEN
```

```
                RAISE EXCEPTION 'To make a bid on this auction
the bidder must have a balance greater than bid value';
ELSE IF EXISTS (SELECT * FROM User WHERE NEW.bidder = id AND
NEW.value <= balance)="" then="" update="" user="" set=""
balance="(SELECT" from="" where="" new.bidder="id)" -=""
new.value="" id="NEW.bidder;" end="" if;="" return="" new;=""
end$$;=""
-- installed as --
CREATE TRIGGER bidder_has_money
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE PROCEDURE bidder_has_money();
```

Trigger reference	TRIGGER04
Trigger description	When the rating of an auction is updated, the rating of its respective creator is also updated with the media of all rated auctions created by this user.

```
CREATE FUNCTION update_ratings() RETURNS trigger
    LANGUAGE plpgsql
    AS $$DECLARE sumOfExistRates integer;
DECLARE countOfExistRates integer;
BEGIN
     IF NEW.rate IS NOT NULL
     THEN
       sumOfExistRates := (SELECT SUM(rate) FROM "auction"
where auctioncreator = NEW.auctioncreator AND rate IS NOT NULL)
;
```

```
       countOfExistRates := (SELECT COUNT(rate) FROM "auction"
where auctioncreator = NEW.auctioncreator AND rate IS NOT NULL)
;
        IF countOfExistRates IS NOT NULL
        THEN UPDATE "user" SET "/rating" = ((sumOfExistRates +
NEW.rate) / (countOfExistRates + 1)) WHERE NEW.auctioncreator =
id;
        ELSE UPDATE "user" SET "/rating" = NEW.rate;
        END IF;
     END IF;
   RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER update_rating
BEFORE INSERT OR UPDATE ON auction
FOR EACH ROW
EXECUTE PROCEDURE update_ratings();
```

Trigger reference	TRIGGER05
Trigger description	If bid is buy now it terminates auction.

```
CREATE FUNCTION buy_now() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF EXISTS (SELECT * FROM "auction" WHERE NEW.auctionbidded = id
AND NEW.value = buyNow AND NEW."isBuyNow" = true)
THEN
UPDATE auction SET state = 'Over'::auctionstate, finaldate =
NEW.date, finalprice = NEW.value, auctionwinner = NEW.bidder
```

```
WHERE id = NEW.auctionbidded;
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'You win this auction!', 'Won
Auction', NEW.auctionbidded, (SELECT bidder FROM "bid" WHERE
auctionbidded = NEW.auctionbidded ORDER BY value DESC LIMIT
1));
END IF;
RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER buy_now
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE PROCEDURE buy_now();
```

Trigger reference	TRIGGER06
Trigger description	A Bid has to have a greater value than starting price of auction.

```
CREATE FUNCTION check_bid_value() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF EXISTS (SELECT * FROM "auction" WHERE NEW.auctionbidded = id
AND NEW.value < startingprice)
THEN
RAISE EXCEPTION 'A Bid has to have a greater value than
starting price of auction.';
END IF;
```

```
RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER check_bid_value
BEFORE INSERT ON bid
FOR EACH ROW
EXECUTE PROCEDURE check_bid_value();
```

Trigger reference	TRIGGER07
Trigger description	If current date is greater than limit date it updates auction to state over and the highest bidder wins the auction.

```
CREATE FUNCTION win_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF (transaction_timestamp() >= NEW.limitdate AND NEW.state =
'Active'::auctionstate)
THEN
UPDATE "auction" SET state = 'Over'::auctionstate, finaldate =
(  SELECT date
  FROM "bid"
  WHERE "bid".auctionBidded= NEW.id
  ORDER BY value DESC  ), finalprice = (  SELECT value
  FROM "bid"
  WHERE "bid".auctionBidded= NEW.id
  ORDER BY value DESC  ), auctionwinner = (  SELECT bidder
  FROM "bid"
  WHERE "bid".auctionBidded=NEW.id
```

```
  ORDER BY value DESC LIMIT 1) WHERE id = NEW.id;
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'You win this auction!', 'Won
Auction', NEW.auctionbidded, (SELECT bidder FROM "bid" WHERE
auctionbidded = NEW.auctionbidded ORDER BY value DESC LIMIT
1));
END IF;
RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER win_auction
BEFORE UPDATE ON auction
FOR EACH ROW
EXECUTE PROCEDURE win_auction();
```

Trigger reference	TRIGGER08
Trigger description	Only the winner of an auction can rate it.

```
-- note that session has to have a current user for this to
happen! That is being created on login --


CREATE FUNCTION winner_rate_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$DECLARE curr_user text;
BEGIN
curr_user := get_current_user();
```

```
IF (NEW.auctionwinner != (SELECT id FROM "user" WHERE
"user".username = curr_user))
THEN RAISE EXCEPTION 'A rate can only be attributed to the
auction by its winner.';
END IF;
RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER winner_rate_auction
BEFORE UPDATE OF rate ON auction
FOR EACH ROW EXECUTE
PROCEDURE winner_rate_auction();
```

Trigger reference	TRIGGER09
Trigger description	Report of an auction generates notification.

```
CREATE FUNCTION auction_reported() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your auction was reported!', 'Auction
Reported', NEW.auctionid, (SELECT "auctioncreator" FROM
"auction" WHERE id = NEW.auctionid));
RETURN NEW;
END;$$;


-- installed as --
```

```
CREATE TRIGGER auction_reported
BEFORE INSERT ON report
FOR EACH ROW
EXECUTE PROCEDURE auction_reported();
```

Trigger reference	TRIGGER10
Trigger description	State Modification generates a notification to user creator.

```
CREATE FUNCTION notification_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF NEW.state = 'Pending'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was created!', 'Auction
Created', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Active'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was accepted!', 'Auction
Accepted', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Rejected'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was rejected!', 'Auction
Rejected', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Over'::auctionstate
THEN
```

```
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction terminated!', 'Auction
Over', NEW.id, NEW.auctioncreator);
END IF;
END IF;
END IF;
END IF;
RETURN NEW;
END;$$;


-- installed as --
CREATE TRIGGER notification_auction
BEFORE INSERT OR UPDATE OF state ON auction
FOR EACH ROW
EXECUTE PROCEDURE notification_auction();
```

Trigger reference	TRIGGER11
Trigger description	On deletion of a comment, a notification is generated for the user that made the comment.

```
CREATE FUNCTION delete_comment() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Comment on this auction was
removed!', 'Comment Removed', OLD.auctioncommented,
OLD.usercommenter);
```

```
RETURN OLD;
END;$$;


-- installed as --
CREATE TRIGGER delete_comment
BEFORE DELETE ON comment
FOR EACH ROW
EXECUTE PROCEDURE delete_comment();
```

Trigger reference	TRIGGER12
Trigger description	On blocking a user, the blocked state of user is put to true or false.

```
CREATE FUNCTION block_user() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF NEW.state = 'Allowed'::blockingstate
THEN UPDATE "user" SET "blocked" = true;
ELSE IF NEW.state = 'Blocked'::blockingstate
THEN UPDATE "user" SET "blocked" = false;
END IF;
END IF;
END;$$;


-- installed as --
CREATE TRIGGER block_user
BEFORE INSERT OR UPDATE ON blocks
```

```
  FOR EACH ROW
  EXECUTE PROCEDURE block_user();
```

4. Complete SQL Code

> The database script must also include the SQL to populate a database with test data with an amount of tuples suitable for testing and with plausible values for the fields of the database.
> This code should also be included in the group's github repository as an SQL script, and a link include here.

> Database File

```
--
-- PostgreSQL database
--

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

SET search_path = public, pg_catalog;

ALTER TABLE ONLY public.report DROP CONSTRAINT
report_normaluserid_fkey;
ALTER TABLE ONLY public.report DROP CONSTRAINT
report_auctionid_fkey;
ALTER TABLE ONLY public.notification DROP CONSTRAINT
notification_authenticated_userid_fkey;
ALTER TABLE ONLY public.notification DROP CONSTRAINT
notification_auctionassociated_fkey;
```

```
ALTER TABLE ONLY public.edit_moderator DROP CONSTRAINT
edit_moderator_removeradmin_fkey;
ALTER TABLE ONLY public.edit_moderator DROP CONSTRAINT
edit_moderator_removedmod_fkey;
ALTER TABLE ONLY public.edit_categories DROP CONSTRAINT
edit_categories_category_fkey;
ALTER TABLE ONLY public.edit_categories DROP CONSTRAINT
edit_categories_admin_fkey;
ALTER TABLE ONLY public.comment DROP CONSTRAINT
comment_usercommenter_fkey;
ALTER TABLE ONLY public.comment DROP CONSTRAINT
comment_auctioncommented_fkey;
ALTER TABLE ONLY public.city DROP CONSTRAINT
city_country_fkey;
ALTER TABLE ONLY public.categoryofauction DROP CONSTRAINT
categoryofauction_category_fkey;
ALTER TABLE ONLY public.categoryofauction DROP CONSTRAINT
categoryofauction_auction_fkey;
ALTER TABLE ONLY public.category DROP CONSTRAINT
category_parent_fkey;
ALTER TABLE ONLY public.blocks DROP CONSTRAINT
blocks_blocker_fkey;
ALTER TABLE ONLY public.blocks DROP CONSTRAINT
blocks_blocked_fkey;
ALTER TABLE ONLY public.bid DROP CONSTRAINT
bid_bidder_fkey;
ALTER TABLE ONLY public.bid DROP CONSTRAINT
bid_auctionbidded_fkey;
ALTER TABLE ONLY public."user" DROP CONSTRAINT
authenticated_user_city_fkey;
ALTER TABLE ONLY public.auction DROP CONSTRAINT
auction_responsiblemoderator_fkey;
ALTER TABLE ONLY public.auction DROP CONSTRAINT
auction_auctionwinner_fkey;
ALTER TABLE ONLY public.auction DROP CONSTRAINT
auction_auctioncreator_fkey;
ALTER TABLE ONLY public.add_credits DROP CONSTRAINT
add_credits_user_fkey;
DROP TRIGGER winner_rate_auction ON public.auction;
```

```
DROP TRIGGER win_auction ON public.auction;
DROP TRIGGER update_rating ON public.auction;
DROP TRIGGER notification_auction ON public.auction;
DROP TRIGGER delete_comment ON public.comment;
DROP TRIGGER check_bid_value ON public.bid;
DROP TRIGGER buy_now ON public.bid;
DROP TRIGGER block_user ON public.blocks;
DROP TRIGGER bidder_has_money ON public.bid;
DROP TRIGGER bid_greater_than_last ON public.bid;
DROP TRIGGER auction_reported ON public.report;
DROP TRIGGER auction_creator ON public.bid;
DROP TRIGGER add_credits_trigger ON public.add_credits;
ALTER TABLE ONLY public.report DROP CONSTRAINT
report_pkey;
ALTER TABLE ONLY public.notification DROP CONSTRAINT
notification_pkey;
ALTER TABLE ONLY public.edit_moderator DROP CONSTRAINT
edit_moderator_pkey;
ALTER TABLE ONLY public.edit_categories DROP CONSTRAINT
edit_categories_pkey;
ALTER TABLE ONLY public.country DROP CONSTRAINT
country_pkey;
ALTER TABLE ONLY public.country DROP CONSTRAINT
country_name_key;
ALTER TABLE ONLY public.comment DROP CONSTRAINT
comment_pkey;
ALTER TABLE ONLY public.city DROP CONSTRAINT city_pkey;
ALTER TABLE ONLY public.city DROP CONSTRAINT
city_name_key;
ALTER TABLE ONLY public.categoryofauction DROP CONSTRAINT
categoryofauction_pkey;
ALTER TABLE ONLY public.category DROP CONSTRAINT
category_pkey;
ALTER TABLE ONLY public.category DROP CONSTRAINT
category_name_key;
ALTER TABLE ONLY public.blocks DROP CONSTRAINT
blocks_pkey;
ALTER TABLE ONLY public.bid DROP CONSTRAINT bid_pkey;
ALTER TABLE ONLY public."user" DROP CONSTRAINT
```

```
authenticated_user_username_key;
ALTER TABLE ONLY public."user" DROP CONSTRAINT
authenticated_user_pkey;
ALTER TABLE ONLY public."user" DROP CONSTRAINT
authenticated_user_email_key;
ALTER TABLE ONLY public.auction DROP CONSTRAINT
auction_pkey;
ALTER TABLE ONLY public.add_credits DROP CONSTRAINT
add_credits_pkey;
DROP TABLE public."user";
DROP TABLE public.report;
DROP TABLE public.notification;
DROP TABLE public.edit_moderator;
DROP TABLE public.edit_categories;
DROP TABLE public.country;
DROP TABLE public.comment;
DROP TABLE public.city;
DROP TABLE public.categoryofauction;
DROP TABLE public.category;
DROP TABLE public.blocks;
DROP TABLE public.bid;
DROP SEQUENCE public.auto_increment_user;
DROP SEQUENCE public.auto_increment_notification;
DROP SEQUENCE public.auto_increment_comment;
DROP SEQUENCE public.auto_increment_city;
DROP SEQUENCE public.auto_increment_category;
DROP TABLE public.auction;
DROP SEQUENCE public.auto_increment_auction;
DROP TABLE public.add_credits;
DROP SEQUENCE public.auto_increment_credits;
DROP FUNCTION public.winner_rate_auction();
DROP FUNCTION public.win_auction();
DROP FUNCTION public.update_ratings();
DROP FUNCTION public.notification_auction();
DROP FUNCTION public.get_current_user();
DROP FUNCTION public.delete_comment();
DROP FUNCTION public.check_rejected_auction(state
auctionstate, dateofrefusal timestamp with time zone,
reasonofrefusal character varying);
```

```
DROP FUNCTION public.check_edit_moderators(removedmod
integer, removeradmin integer);
DROP FUNCTION public.check_block_users(blocked integer,
blocker integer);
DROP FUNCTION public.check_bid_value();
DROP FUNCTION public.check_auction_win(state
auctionstate, finaldate timestamp with time zone,
finalprice integer, auctionwinner integer);
DROP FUNCTION public.check_auction_users(auctioncreator
integer, auctionwinner integer, responsiblemoderator
integer);
DROP FUNCTION public.check_admin_modify_category(admin
integer);
DROP FUNCTION public.buy_now();
DROP FUNCTION public.block_user();
DROP FUNCTION public.bidder_has_money();
DROP FUNCTION public.bid_greater_than_last();
DROP FUNCTION public.auction_reported();
DROP FUNCTION public.auction_creator();
DROP FUNCTION public.add_credits_trigger();
DROP TYPE public.typeofuser;
DROP TYPE public.blockingstate;
DROP TYPE public.auctionstate;
DROP SCHEMA public;
--
-- Name: public; Type: SCHEMA; Schema: -; Owner: lbaw1716
--

CREATE SCHEMA public;


ALTER SCHEMA public OWNER TO lbaw1716;

--
-- Name: SCHEMA public; Type: COMMENT; Schema: -; Owner:
lbaw1716
--

COMMENT ON SCHEMA public IS 'standard public schema';
```

```
SET search_path = public, pg_catalog;

--
-- Name: auctionstate; Type: TYPE; Schema: public; Owner:
lbaw1716
--

CREATE TYPE auctionstate AS ENUM (
    'Active',
    'Rejected',
    'Pending',
    'Over'
);


ALTER TYPE auctionstate OWNER TO lbaw1716;

--
-- Name: blockingstate; Type: TYPE; Schema: public;
Owner: lbaw1716
--

CREATE TYPE blockingstate AS ENUM (
    'Blocked',
    'Allowed'
);


ALTER TYPE blockingstate OWNER TO lbaw1716;

--
-- Name: typeofuser; Type: TYPE; Schema: public; Owner:
lbaw1716
--

CREATE TYPE typeofuser AS ENUM (
    'Moderator',
```

```
    'Administrator',
    'Normal'
);


ALTER TYPE typeofuser OWNER TO lbaw1716;

--
-- Name: add_credits_trigger(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION add_credits_trigger() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
UPDATE "user" SET balance = ((SELECT balance FROM "user"
WHERE id = NEW.user) +  NEW.value) WHERE id = NEW.user;
RETURN NEW;
END;$$;


ALTER FUNCTION public.add_credits_trigger() OWNER TO
lbaw1716;

--
-- Name: auction_creator(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION auction_creator() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
  IF EXISTS (SELECT * FROM auction
    INNER JOIN "user" ON "user".id =
auction.auctioncreator
    WHERE NEW.auctionbidded = auction.id AND NEW.bidder =
"user".id) THEN
    RAISE EXCEPTION 'The creator of an auction cannot
make a bid on it!';
```

```
  END IF;
  RETURN NEW;
END;$$;


ALTER FUNCTION public.auction_creator() OWNER TO
lbaw1716;

--
-- Name: auction_reported(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION auction_reported() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your auction was reported!',
'Auction Reported', NEW.auctionid, (SELECT
"auctioncreator" FROM "auction" WHERE id =
NEW.auctionid));
RETURN NEW;
END;$$;


ALTER FUNCTION public.auction_reported() OWNER TO
lbaw1716;

--
-- Name: bid_greater_than_last(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION bid_greater_than_last() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
        IF EXISTS (SELECT * FROM bid WHERE
NEW.auctionbidded = auctionbidded AND NEW.value <= value)
```

```
THEN
                RAISE EXCEPTION 'A bid on this auction
has to have a greater value than a previous one on this
auction.';
        ELSE
         UPDATE "user" SET balance = ((SELECT value FROM
"bid" WHERE "bid".auctionBidded=NEW.auctionbidded ORDER
BY value DESC LIMIT 1) + (SELECT balance FROM "user"
WHERE id = (SELECT bidder FROM "bid" WHERE auctionbidded
= NEW.auctionbidded ORDER BY value DESC LIMIT 1))) WHERE
id = (SELECT bidder FROM "bid" WHERE auctionbidded =
NEW.auctionbidded ORDER BY value DESC LIMIT 1);
        INSERT INTO Notification (id, date, description,
type, auctionassociated, authenticated_userid) VALUES
(DEFAULT, transaction_timestamp(), 'Your bid on this
auction was surpassed. Try again!', 'Bid Exceeded',
NEW.auctionbidded, (SELECT bidder FROM "bid" WHERE
auctionbidded = NEW.auctionbidded ORDER BY value DESC
LIMIT 1));
        END IF;
        RETURN NEW;
END$$;


ALTER FUNCTION public.bid_greater_than_last() OWNER TO
lbaw1716;

--
-- Name: FUNCTION bid_greater_than_last(); Type: COMMENT;
Schema: public; Owner: lbaw1716
--

COMMENT ON FUNCTION bid_greater_than_last() IS '       ';



--
-- Name: bidder_has_money(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--
```

```
CREATE FUNCTION bidder_has_money() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
        IF EXISTS (SELECT * FROM "user" WHERE NEW.bidder
= id AND NEW.value > balance)          THEN
                RAISE EXCEPTION 'To make a bid on this
auction the bidder must have a balance greater than bid
value';
ELSE IF EXISTS (SELECT * FROM "user" WHERE NEW.bidder =
id AND NEW.value <= balance)
THEN
      UPDATE "user" SET balance = (SELECT balance FROM
"user" WHERE NEW.bidder = id) - NEW.value WHERE id =
NEW.bidder;
END IF;
        END IF;
        RETURN NEW;
END$$;


ALTER FUNCTION public.bidder_has_money() OWNER TO
lbaw1716;

--
-- Name: block_user(); Type: FUNCTION; Schema: public;
Owner: lbaw1716
--

CREATE FUNCTION block_user() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF NEW.state = 'Allowed'::blockingstate
THEN UPDATE "user" SET "blocked" = true;
ELSE IF NEW.state = 'Blocked'::blockingstate
THEN UPDATE "user" SET "blocked" = false;
END IF;
END IF;
END;$$;
```

```
ALTER FUNCTION public.block_user() OWNER TO lbaw1716;

--
-- Name: buy_now(); Type: FUNCTION; Schema: public;
Owner: lbaw1716
--

CREATE FUNCTION buy_now() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF EXISTS (SELECT * FROM "auction" WHERE
NEW.auctionbidded = id AND NEW.value = buyNow AND
NEW."isBuyNow" = true)
THEN
UPDATE auction SET state = 'Over'::auctionstate,
finaldate = NEW.date, finalprice = NEW.value,
auctionwinner = NEW.bidder WHERE id = NEW.auctionbidded;
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'You win this auction!', 'Won
Auction', NEW.auctionbidded, (SELECT bidder FROM "bid"
WHERE auctionbidded = NEW.auctionbidded ORDER BY value
DESC LIMIT 1));
END IF;
RETURN NEW;
END;$$;


ALTER FUNCTION public.buy_now() OWNER TO lbaw1716;

--
-- Name: check_admin_modify_category(integer); Type:
FUNCTION; Schema: public; Owner: lbaw1716
--

CREATE FUNCTION check_admin_modify_category(admin
integer) RETURNS boolean
```

```
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE adminUser TypeOfUser;
BEGIN
SELECT typeOfUser INTO adminUser FROM "User" where id =
admin ;
IF
adminUser = 'Administrator'::TypeOfUser THEN passed :=
true; ELSE passed := false;
END IF;
RETURN passed;
END;
$$;


ALTER FUNCTION public.check_admin_modify_category(admin
integer) OWNER TO lbaw1716;

--
-- Name: check_auction_users(integer, integer, integer);
Type: FUNCTION; Schema: public; Owner: lbaw1716
--

CREATE FUNCTION check_auction_users(auctioncreator
integer, auctionwinner integer, responsiblemoderator
integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE createrUser TypeOfUser;
DECLARE winnerUser TypeOfUser;
DECLARE responsibleUser TypeOfUser;
BEGIN
        SELECT typeOfUser INTO createrUser FROM "user"
where id = auctioncreator ;
        IF(auctionwinner IS NULL)
           THEN winnerUser := NULL ;
           ELSE SELECT typeOfUser INTO winnerUser FROM
"user" where id = auctionwinner ;
        END IF;
```

```
        IF(responsiblemoderator IS NULL)
           THEN responsibleUser := NULL ;
           ELSE SELECT typeOfUser INTO responsibleUser
FROM "user" where id = responsiblemoderator ;
        END IF;
        IF createrUser = 'Normal'::TypeOfUser AND
(winnerUser = 'Normal'::TypeOfUser OR winnerUser IS NULL)
AND (
responsibleUser = 'Moderator'::TypeOfUser OR
responsibleUser = 'Administrator'::TypeOfUser OR
responsibleUser IS NULL
)
                THEN passed := true;
                ELSE passed := false;
                END IF;
                RETURN passed;
END;$$;


ALTER FUNCTION public.check_auction_users(auctioncreator
integer, auctionwinner integer, responsiblemoderator
integer) OWNER TO lbaw1716;

--
-- Name: check_auction_win(auctionstate, timestamp with
time zone, integer, integer); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION check_auction_win(state auctionstate,
finaldate timestamp with time zone, finalprice integer,
auctionwinner integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE pass boolean;
BEGIN
  IF state = 'Over'::auctionstate
      THEN IF finaldate IS NOT NULL AND finalprice IS NOT
NULL AND auctionwinner IS NOT NULL
           THEN pass := true;
```

```
            ELSE pass := false;
            END IF;
  ELSE pass := true;
  END IF;
  RETURN pass;
END;$$;


ALTER FUNCTION public.check_auction_win(state
auctionstate, finaldate timestamp with time zone,
finalprice integer, auctionwinner integer) OWNER TO
lbaw1716;

--
-- Name: check_bid_value(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION check_bid_value() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF EXISTS (SELECT * FROM "auction" WHERE
NEW.auctionbidded = id AND NEW.value < startingprice)
THEN
RAISE EXCEPTION 'A Bid has to have a greater value than
starting price of auction.';
END IF;
RETURN NEW;
END;$$;


ALTER FUNCTION public.check_bid_value() OWNER TO
lbaw1716;

--
-- Name: check_block_users(integer, integer); Type:
FUNCTION; Schema: public; Owner: lbaw1716
--
```

```
CREATE FUNCTION check_block_users(blocked integer,
blocker integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed BOOLEAN;
DECLARE blokedUser TypeOfUser;
DECLARE blokerUser TypeOfUser;
BEGIN
                SELECT typeOfUser INTO blokedUser FROM
"User" where id = blocked ;
                SELECT typeOfUser INTO blokerUser FROM
"User" where id = blocker ;
                IF blokedUser = 'Normal'::TypeOfUser AND
(blokerUser = 'Administrator'::TypeOfUser OR blokerUser =
'Moderator'::TypeOfUser)
                THEN passed := true;
                ELSE passed := false;
                END IF;
RETURN passed;
END;$$;


ALTER FUNCTION public.check_block_users(blocked integer,
blocker integer) OWNER TO lbaw1716;

--
-- Name: check_edit_moderators(integer, integer); Type:
FUNCTION; Schema: public; Owner: lbaw1716
--

CREATE FUNCTION check_edit_moderators(removedmod integer,
removeradmin integer) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE passed boolean;
DECLARE removerModUser TypeOfUser;
DECLARE removerAdminUser TypeOfUser;
BEGIN
        SELECT typeOfUser INTO removerModUser FROM "User"
where id = removedMod ;
        SELECT typeOfUser INTO removerAdminUser FROM
```

```
"User" where id = removerAdmin ;
        IF removerModUser = 'Moderator'::TypeOfUser AND
removerAdminUser = 'Administrator'::TypeOfUser
        THEN passed := true;
        ELSE passed := false;
        END IF;
RETURN passed;
END;$$;


ALTER FUNCTION public.check_edit_moderators(removedmod
integer, removeradmin integer) OWNER TO lbaw1716;

--
-- Name: check_rejected_auction(auctionstate, timestamp
with time zone, character varying); Type: FUNCTION;
Schema: public; Owner: lbaw1716
--

CREATE FUNCTION check_rejected_auction(state
auctionstate, dateofrefusal timestamp with time zone,
reasonofrefusal character varying) RETURNS boolean
    LANGUAGE plpgsql
    AS $$DECLARE pass boolean;
BEGIN
  IF state = 'Rejected'::auctionstate
      THEN IF dateOfRefusal IS NOT NULL AND
reasonOfRefusal IS NOT NULL
           THEN pass := true;
           ELSE pass := false;
           END IF;
  ELSE pass := true;
  END IF;
  RETURN pass;
END;$$;


ALTER FUNCTION public.check_rejected_auction(state
auctionstate, dateofrefusal timestamp with time zone,
```

```
reasonofrefusal character varying) OWNER TO lbaw1716;

--
-- Name: delete_comment(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION delete_comment() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Comment on this auction
was removed!', 'Comment Removed', OLD.auctioncommented,
OLD.usercommenter);
RETURN OLD;
END;$$;


ALTER FUNCTION public.delete_comment() OWNER TO lbaw1716;

--
-- Name: get_current_user(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION get_current_user() RETURNS text
    LANGUAGE plpgsql
    AS $$DECLARE
    cur_user varchar;
BEGIN
    BEGIN
        cur_user := (SELECT "username" FROM curr_user);
    EXCEPTION WHEN undefined_table THEN
        cur_user := 'unknown_user';
    END;
    RETURN cur_user;
END;
$$;
```

```
ALTER FUNCTION public.get_current_user() OWNER TO
lbaw1716;

--
-- Name: notification_auction(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION notification_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF NEW.state = 'Pending'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was created!',
'Auction Created', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Active'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was accepted!',
'Auction Accepted', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Rejected'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction was rejected!',
'Auction Rejected', NEW.id, NEW.auctioncreator);
ELSE IF NEW.state = 'Over'::auctionstate
THEN
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'Your Auction terminated!',
'Auction Over', NEW.id, NEW.auctioncreator);
END IF;
END IF;
```

```
END IF;
END IF;
RETURN NEW;
END;$$;


ALTER FUNCTION public.notification_auction() OWNER TO
lbaw1716;

--
-- Name: update_ratings(); Type: FUNCTION; Schema:
public; Owner: lbaw1716
--

CREATE FUNCTION update_ratings() RETURNS trigger
    LANGUAGE plpgsql
    AS $$DECLARE sumOfExistRates integer;
DECLARE countOfExistRates integer;
BEGIN
     IF NEW.rate IS NOT NULL
     THEN
       sumOfExistRates := (SELECT SUM(rate) FROM
"auction" where auctioncreator = NEW.auctioncreator AND
rate IS NOT NULL) ;
       countOfExistRates := (SELECT COUNT(rate) FROM
"auction" where auctioncreator = NEW.auctioncreator AND
rate IS NOT NULL) ;
        IF countOfExistRates IS NOT NULL
        THEN UPDATE "user" SET "/rating" =
((sumOfExistRates + NEW.rate) / (countOfExistRates + 1))
WHERE NEW.auctioncreator = id;
        ELSE UPDATE "user" SET "/rating" = NEW.rate;
        END IF;
     END IF;
   RETURN NEW;
END;$$;


ALTER FUNCTION public.update_ratings() OWNER TO lbaw1716;
```

```
--
-- Name: win_auction(); Type: FUNCTION; Schema: public;
Owner: lbaw1716
--

CREATE FUNCTION win_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$BEGIN
IF (transaction_timestamp() >= NEW.limitdate AND
NEW.state = 'Active'::auctionstate)
THEN
UPDATE "auction" SET state = 'Over'::auctionstate,
finaldate = (  SELECT date
  FROM "bid"
  WHERE "bid".auctionBidded= NEW.id
  ORDER BY value DESC  ), finalprice = (  SELECT value
  FROM "bid"
  WHERE "bid".auctionBidded= NEW.id
  ORDER BY value DESC  ), auctionwinner = (  SELECT
bidder
  FROM "bid"
  WHERE "bid".auctionBidded=NEW.id
  ORDER BY value DESC LIMIT 1) WHERE id = NEW.id;
INSERT INTO Notification (id, date, description, type,
auctionassociated, authenticated_userid) VALUES (DEFAULT,
transaction_timestamp(), 'You win this auction!', 'Won
Auction', NEW.auctionbidded, (SELECT bidder FROM "bid"
WHERE auctionbidded = NEW.auctionbidded ORDER BY value
DESC LIMIT 1));
END IF;
RETURN NEW;
END;$$;


ALTER FUNCTION public.win_auction() OWNER TO lbaw1716;

--
-- Name: winner_rate_auction(); Type: FUNCTION; Schema:
```

```
public; Owner: lbaw1716
--

CREATE FUNCTION winner_rate_auction() RETURNS trigger
    LANGUAGE plpgsql
    AS $$DECLARE curr_user text;
BEGIN
curr_user := get_current_user();
IF (NEW.auctionwinner != (SELECT id FROM "user" WHERE
"user".username = curr_user))
THEN RAISE EXCEPTION 'A rate can only be attributed to
the auction by its winner.';
END IF;
RETURN NEW;
END;$$;


ALTER FUNCTION public.winner_rate_auction() OWNER TO
lbaw1716;

--
-- Name: auto_increment_credits; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_credits
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_credits OWNER TO lbaw1716;

SET default_tablespace = '';

SET default_with_oids = false;
```

```
--
-- Name: add_credits; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE add_credits (
    id integer DEFAULT
nextval('auto_increment_credits'::regclass) NOT NULL,
    value integer NOT NULL,
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    paypalid character varying NOT NULL,
    "user" integer NOT NULL,
    "trasactionID" character varying NOT NULL
);


ALTER TABLE add_credits OWNER TO lbaw1716;

--
-- Name: auto_increment_auction; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_auction
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_auction OWNER TO lbaw1716;

--
-- Name: auction; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--
```

```
CREATE TABLE auction (
    id integer DEFAULT
nextval('auto_increment_auction'::regclass) NOT NULL,
    state auctionstate NOT NULL,
    title character varying NOT NULL,
    description character varying,
    sellingreason character varying,
    pathtophoto character varying,
    startingprice integer NOT NULL,
    minimumsellingprice integer,
    buynow integer,
    startdate timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    limitdate timestamp with time zone NOT NULL,
    refusaldate timestamp with time zone,
    "/numberOfBids" integer,
    reasonofrefusal character varying,
    finaldate timestamp with time zone,
    finalprice integer,
    rate integer,
    auctioncreator integer NOT NULL,
    auctionwinner integer,
    responsiblemoderator integer,
    CONSTRAINT auction_buynow_check CHECK ((buynow > 0)),
    CONSTRAINT auction_check CHECK
((((minimumsellingprice > startingprice) OR
(minimumsellingprice = NULL::integer)) AND ((buynow >
startingprice) OR (buynow = NULL::integer)))),
    CONSTRAINT auction_minimumsellingprice_check CHECK
((minimumsellingprice > 0)),
    CONSTRAINT auction_rate_check CHECK (((rate >= 0) AND
(rate <= 5))),
    CONSTRAINT auction_startingprice_check CHECK
((startingprice > 0)),
    CONSTRAINT check_auction_win CHECK
(check_auction_win(state, finaldate, finalprice,
auctionwinner)),
    CONSTRAINT check_dates CHECK ((age(limitdate,
startdate) <= '7 days'::interval)),
```

```
    CONSTRAINT check_rejected CHECK
(check_rejected_auction(state, refusaldate,
reasonofrefusal)),
    CONSTRAINT check_users CHECK
(check_auction_users(auctioncreator, auctionwinner,
responsiblemoderator))
);


ALTER TABLE auction OWNER TO lbaw1716;

--
-- Name: auto_increment_category; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_category
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_category OWNER TO lbaw1716;

--
-- Name: auto_increment_city; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_city
    START WITH 4
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

```
ALTER TABLE auto_increment_city OWNER TO lbaw1716;

--
-- Name: auto_increment_comment; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_comment
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_comment OWNER TO lbaw1716;

--
-- Name: auto_increment_notification; Type: SEQUENCE;
Schema: public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_notification
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_notification OWNER TO
lbaw1716;

--
-- Name: auto_increment_user; Type: SEQUENCE; Schema:
public; Owner: lbaw1716
--

CREATE SEQUENCE auto_increment_user
```

```
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE auto_increment_user OWNER TO lbaw1716;

--
-- Name: bid; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE bid (
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    value integer NOT NULL,
    auctionbidded integer NOT NULL,
    bidder integer NOT NULL,
    "isBuyNow" boolean
);


ALTER TABLE bid OWNER TO lbaw1716;

--
-- Name: blocks; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE blocks (
    state blockingstate NOT NULL,
    description character varying,
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    blocked integer NOT NULL,
    blocker integer NOT NULL,
    CONSTRAINT check_users CHECK
```

```
(check_block_users(blocked, blocker))
);


ALTER TABLE blocks OWNER TO lbaw1716;

--
-- Name: category; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE category (
    categoryid integer DEFAULT
nextval('auto_increment_category'::regclass) NOT NULL,
    name character varying(50) NOT NULL,
    parent integer,
    CONSTRAINT check_parent_not_equal CHECK ((parent <>
categoryid))
);


ALTER TABLE category OWNER TO lbaw1716;

--
-- Name: categoryofauction; Type: TABLE; Schema: public;
Owner: lbaw1716; Tablespace:
--

CREATE TABLE categoryofauction (
    category integer NOT NULL,
    auction integer NOT NULL
);


ALTER TABLE categoryofauction OWNER TO lbaw1716;

--
-- Name: city; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
```

```
--

CREATE TABLE city (
    id integer DEFAULT
nextval('auto_increment_city'::regclass) NOT NULL,
    name character varying(50) NOT NULL,
    country integer NOT NULL
);


ALTER TABLE city OWNER TO lbaw1716;

--
-- Name: comment; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE comment (
    id integer DEFAULT
nextval('auto_increment_comment'::regclass) NOT NULL,
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    description character varying NOT NULL,
    auctioncommented integer NOT NULL,
    usercommenter integer NOT NULL
);


ALTER TABLE comment OWNER TO lbaw1716;

--
-- Name: country; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE country (
    id integer DEFAULT
nextval('auto_increment_auction'::regclass) NOT NULL,
    name character varying(50) NOT NULL
```

```
);


ALTER TABLE country OWNER TO lbaw1716;

--
-- Name: edit_categories; Type: TABLE; Schema: public;
Owner: lbaw1716; Tablespace:
--

CREATE TABLE edit_categories (
    category integer NOT NULL,
    admin integer NOT NULL,
    CONSTRAINT check_admin_modify_category CHECK
(check_admin_modify_category(admin))
);


ALTER TABLE edit_categories OWNER TO lbaw1716;

--
-- Name: edit_moderator; Type: TABLE; Schema: public;
Owner: lbaw1716; Tablespace:
--

CREATE TABLE edit_moderator (
    removedmod integer NOT NULL,
    removeradmin integer NOT NULL,
    CONSTRAINT check_users CHECK
(check_edit_moderators(removedmod, removeradmin))
);


ALTER TABLE edit_moderator OWNER TO lbaw1716;

--
-- Name: notification; Type: TABLE; Schema: public;
Owner: lbaw1716; Tablespace:
--
```

```
CREATE TABLE notification (
    id integer DEFAULT
nextval('auto_increment_notification'::regclass) NOT
NULL,
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    description character varying(50) NOT NULL,
    type character varying(50) NOT NULL,
    auctionassociated integer,
    authenticated_userid integer NOT NULL,
    read boolean DEFAULT false
);


ALTER TABLE notification OWNER TO lbaw1716;

--
-- Name: report; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE report (
    date timestamp with time zone DEFAULT
transaction_timestamp() NOT NULL,
    reason character varying NOT NULL,
    auctionid integer NOT NULL,
    normaluserid integer NOT NULL
);


ALTER TABLE report OWNER TO lbaw1716;

--
-- Name: user; Type: TABLE; Schema: public; Owner:
lbaw1716; Tablespace:
--

CREATE TABLE "user" (
```

```
    id integer DEFAULT
nextval('auto_increment_user'::regclass) NOT NULL,
    typeofuser typeofuser NOT NULL,
    username character varying(50) NOT NULL,
    password character varying NOT NULL,
    pathtophoto character varying,
    completename character varying,
    email character varying,
    birthdate date,
    "/rating" integer,
    address character varying,
    postalcode character varying(50),
    balance integer,
    city integer,
    phonenumber numeric,
    blocked boolean DEFAULT false NOT NULL,
    CONSTRAINT "authenticated_user_/rating_check" CHECK
((("/rating" >= 0) AND ("/rating" <= 5))),
    CONSTRAINT authenticated_user_balance_check CHECK
((balance >= 0))
);


ALTER TABLE "user" OWNER TO lbaw1716;

--
-- Name: add_credits_pkey; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY add_credits
    ADD CONSTRAINT add_credits_pkey PRIMARY KEY (id);



--
-- Name: auction_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--
```

```
ALTER TABLE ONLY auction
    ADD CONSTRAINT auction_pkey PRIMARY KEY (id);


--
-- Name: authenticated_user_email_key; Type: CONSTRAINT;
Schema: public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY "user"
    ADD CONSTRAINT authenticated_user_email_key UNIQUE
(email);


--
-- Name: authenticated_user_pkey; Type: CONSTRAINT;
Schema: public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY "user"
    ADD CONSTRAINT authenticated_user_pkey PRIMARY KEY
(id);


--
-- Name: authenticated_user_username_key; Type:
CONSTRAINT; Schema: public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY "user"
    ADD CONSTRAINT authenticated_user_username_key UNIQUE
(username);


--
-- Name: bid_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--
```

```
ALTER TABLE ONLY bid
    ADD CONSTRAINT bid_pkey PRIMARY KEY (date,
auctionbidded, bidder);


--
-- Name: blocks_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY blocks
    ADD CONSTRAINT blocks_pkey PRIMARY KEY (blocked,
blocker);


--
-- Name: category_name_key; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY category
    ADD CONSTRAINT category_name_key UNIQUE (name);


--
-- Name: category_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY category
    ADD CONSTRAINT category_pkey PRIMARY KEY
(categoryid);


--
-- Name: categoryofauction_pkey; Type: CONSTRAINT;
Schema: public; Owner: lbaw1716; Tablespace:
--
```

```
ALTER TABLE ONLY categoryofauction
    ADD CONSTRAINT categoryofauction_pkey PRIMARY KEY
(category, auction);


--
-- Name: city_name_key; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY city
    ADD CONSTRAINT city_name_key UNIQUE (name);


--
-- Name: city_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY city
    ADD CONSTRAINT city_pkey PRIMARY KEY (id);


--
-- Name: comment_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY comment
    ADD CONSTRAINT comment_pkey PRIMARY KEY (id);


--
-- Name: country_name_key; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY country
    ADD CONSTRAINT country_name_key UNIQUE (name);
```

```
--
-- Name: country_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY country
    ADD CONSTRAINT country_pkey PRIMARY KEY (id);



--
-- Name: edit_categories_pkey; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY edit_categories
    ADD CONSTRAINT edit_categories_pkey PRIMARY KEY
(category, admin);



--
-- Name: edit_moderator_pkey; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY edit_moderator
    ADD CONSTRAINT edit_moderator_pkey PRIMARY KEY
(removedmod, removeradmin);



--
-- Name: notification_pkey; Type: CONSTRAINT; Schema:
public; Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY notification
    ADD CONSTRAINT notification_pkey PRIMARY KEY (id);
```

```
--
-- Name: report_pkey; Type: CONSTRAINT; Schema: public;
Owner: lbaw1716; Tablespace:
--

ALTER TABLE ONLY report
    ADD CONSTRAINT report_pkey PRIMARY KEY (auctionid,
normaluserid);


--
-- Name: add_credits_trigger; Type: TRIGGER; Schema:
public; Owner: lbaw1716
--

CREATE TRIGGER add_credits_trigger BEFORE INSERT ON
add_credits FOR EACH ROW EXECUTE PROCEDURE
add_credits_trigger();


--
-- Name: auction_creator; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER auction_creator BEFORE INSERT ON bid FOR
EACH ROW EXECUTE PROCEDURE auction_creator();


--
-- Name: auction_reported; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER auction_reported BEFORE INSERT ON report
FOR EACH ROW EXECUTE PROCEDURE auction_reported();
```

```
--
-- Name: bid_greater_than_last; Type: TRIGGER; Schema:
public; Owner: lbaw1716
--

CREATE TRIGGER bid_greater_than_last BEFORE INSERT ON bid
FOR EACH ROW EXECUTE PROCEDURE bid_greater_than_last();


--
-- Name: bidder_has_money; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER bidder_has_money BEFORE INSERT ON bid FOR
EACH ROW EXECUTE PROCEDURE bidder_has_money();


--
-- Name: block_user; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER block_user BEFORE INSERT OR UPDATE ON
blocks FOR EACH ROW EXECUTE PROCEDURE block_user();

--
-- Name: buy_now; Type: TRIGGER; Schema: public; Owner:
lbaw1716
--

CREATE TRIGGER buy_now BEFORE INSERT ON bid FOR EACH ROW
EXECUTE PROCEDURE buy_now();


--
-- Name: check_bid_value; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--
```

```
CREATE TRIGGER check_bid_value BEFORE INSERT ON bid FOR
EACH ROW EXECUTE PROCEDURE check_bid_value();


--
-- Name: delete_comment; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER delete_comment BEFORE DELETE ON comment
FOR EACH ROW EXECUTE PROCEDURE delete_comment();


--
-- Name: notification_auction; Type: TRIGGER; Schema:
public; Owner: lbaw1716
--

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


--
-- Name: update_rating; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

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


--
-- Name: win_auction; Type: TRIGGER; Schema: public;
Owner: lbaw1716
--

CREATE TRIGGER win_auction BEFORE UPDATE ON auction FOR
```

```
EACH ROW EXECUTE PROCEDURE win_auction();


--
-- Name: winner_rate_auction; Type: TRIGGER; Schema:
public; Owner: lbaw1716
--

CREATE TRIGGER winner_rate_auction BEFORE UPDATE OF rate
ON auction FOR EACH ROW EXECUTE PROCEDURE
winner_rate_auction();


--
-- Name: add_credits_user_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY add_credits
    ADD CONSTRAINT add_credits_user_fkey FOREIGN KEY
("user") REFERENCES "user"(id);


--
-- Name: auction_auctioncreator_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY auction
    ADD CONSTRAINT auction_auctioncreator_fkey FOREIGN
KEY (auctioncreator) REFERENCES "user"(id);


--
-- Name: auction_auctionwinner_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY auction
```

```
    ADD CONSTRAINT auction_auctionwinner_fkey FOREIGN KEY
(auctionwinner) REFERENCES "user"(id);


--
-- Name: auction_responsiblemoderator_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY auction
    ADD CONSTRAINT auction_responsiblemoderator_fkey
FOREIGN KEY (responsiblemoderator) REFERENCES "user"(id);


--
-- Name: authenticated_user_city_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY "user"
    ADD CONSTRAINT authenticated_user_city_fkey FOREIGN
KEY (city) REFERENCES city(id);


--
-- Name: bid_auctionbidded_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY bid
    ADD CONSTRAINT bid_auctionbidded_fkey FOREIGN KEY
(auctionbidded) REFERENCES auction(id);


--
-- Name: bid_bidder_fkey; Type: FK CONSTRAINT; Schema:
public; Owner: lbaw1716
--
```

```
ALTER TABLE ONLY bid
    ADD CONSTRAINT bid_bidder_fkey FOREIGN KEY (bidder)
REFERENCES "user"(id);


--
-- Name: blocks_blocked_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY blocks
    ADD CONSTRAINT blocks_blocked_fkey FOREIGN KEY
(blocked) REFERENCES "user"(id);


--
-- Name: blocks_blocker_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY blocks
    ADD CONSTRAINT blocks_blocker_fkey FOREIGN KEY
(blocker) REFERENCES "user"(id);


--
-- Name: category_parent_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY category
    ADD CONSTRAINT category_parent_fkey FOREIGN KEY
(parent) REFERENCES category(categoryid);


--
-- Name: categoryofauction_auction_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--
```

```
ALTER TABLE ONLY categoryofauction
    ADD CONSTRAINT categoryofauction_auction_fkey FOREIGN
KEY (auction) REFERENCES auction(id);


--
-- Name: categoryofauction_category_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY categoryofauction
    ADD CONSTRAINT categoryofauction_category_fkey
FOREIGN KEY (category) REFERENCES category(categoryid);


--
-- Name: city_country_fkey; Type: FK CONSTRAINT; Schema:
public; Owner: lbaw1716
--

ALTER TABLE ONLY city
    ADD CONSTRAINT city_country_fkey FOREIGN KEY
(country) REFERENCES country(id);


--
-- Name: comment_auctioncommented_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY comment
    ADD CONSTRAINT comment_auctioncommented_fkey FOREIGN
KEY (auctioncommented) REFERENCES auction(id);


--
-- Name: comment_usercommenter_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
```

```
--

ALTER TABLE ONLY comment
    ADD CONSTRAINT comment_usercommenter_fkey FOREIGN KEY
(usercommenter) REFERENCES "user"(id);



--
-- Name: edit_categories_admin_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY edit_categories
    ADD CONSTRAINT edit_categories_admin_fkey FOREIGN KEY
(admin) REFERENCES "user"(id);



--
-- Name: edit_categories_category_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY edit_categories
    ADD CONSTRAINT edit_categories_category_fkey FOREIGN
KEY (category) REFERENCES category(categoryid);



--
-- Name: edit_moderator_removedmod_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY edit_moderator
    ADD CONSTRAINT edit_moderator_removedmod_fkey FOREIGN
KEY (removedmod) REFERENCES "user"(id);



--
-- Name: edit_moderator_removeradmin_fkey; Type: FK
```

```
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY edit_moderator
    ADD CONSTRAINT edit_moderator_removeradmin_fkey
FOREIGN KEY (removeradmin) REFERENCES "user"(id);



--
-- Name: notification_auctionassociated_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY notification
    ADD CONSTRAINT notification_auctionassociated_fkey
FOREIGN KEY (auctionassociated) REFERENCES auction(id);



--
-- Name: notification_authenticated_userid_fkey; Type: FK
CONSTRAINT; Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY notification
    ADD CONSTRAINT notification_authenticated_userid_fkey
FOREIGN KEY (authenticated_userid) REFERENCES "user"(id);



--
-- Name: report_auctionid_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY report
    ADD CONSTRAINT report_auctionid_fkey FOREIGN KEY
(auctionid) REFERENCES auction(id);



--
```

```
-- Name: report_normaluserid_fkey; Type: FK CONSTRAINT;
Schema: public; Owner: lbaw1716
--

ALTER TABLE ONLY report
    ADD CONSTRAINT report_normaluserid_fkey FOREIGN KEY
(normaluserid) REFERENCES "user"(id);


--
-- Name: public; Type: ACL; Schema: -; Owner: lbaw1716
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM lbaw1716;
GRANT ALL ON SCHEMA public TO lbaw1716;
```

> Data File

Changes made to the first submission:

1. Added some more triggers.
2. Changed some of the DELETES and INSERTS.
3. Changed Authenticated_User to User.
4. We reviewed our sql so it was updated in according to suggestions.
5. We took inserts code from this file and put it only on link.
6. We changed some of the indexes as suggested.

GROUP1716, 08/04/2018

- Diogo Peixoto Pereira, diogopeixotopereira@gmail.com
- Igor Bernardo Amorim Silveira, igorasilveira@gmail.com
- Nádia de Sousa Varela de Carvalho, nadiacarvalho118@gmail.com
- Pedro Miguel Ferraz Nogueira da Silva, pedronogueirasilva5@gmail.com