A9: Main Accesses to the database and transactions

This artefact shows the main accesses to the database, including the transaction. For each transaction, the isolation level must be explicitly stated and read-only transactions must be identified to improve global performance. For each identified access, the SQL code and the reference of web resources (A7) must be presented. A given database access can be used in more than one resource.

1 Main Accesses

1.1 M01: Authentication

SQL101	Creates a new user	R105
`INSERT INTO "user" (name, username, hashed_pass, email, short_bio, register_date, profile_pic) VALUES (?, ?, ?, ?, ?, ?, ?)`		
SQL102	Verifies if an user exists	R108
`SELECT * FROM "user" WHERE "user".username = ?`		
SQL103	Verifies if an admin exists	R107
`SELECT * FROM admin WHERE username = ?`		

1.2 M02: Users

SQL201	Get the most recent notifications of an user	This access is always used in the following modules: M02, M03, M04 provided that the user is authenticated

```
SELECT notification.*
FROM notification
INNER JOIN "user" ON notification.user_id = "user".id
WHERE is_new = TRUE AND "user".id = ?
LIMIT 5;
```

SQL202	Get user's notifications	R203

```
SELECT *
FROM notification
WHERE user_id = ?
GROUP BY id, is_new
ORDER BY DATE DESC;
```

SQL203	Remove notification	R203

```
DELETE
FROM notification
WHERE id = ?;
```

SQL204	Read notification	R203

```
UPDATE notification
SET is_new = "false"
WHERE id = ?;
```

SQL205	Get city and country	R201

```
SELECT city.name AS city, country.name AS country
FROM "user"
JOIN location ON "user".location_id = location.id
JOIN city ON location.city_id = city.id
JOIN country ON city.country_id = country.id
WHERE "user".id = :user_id
```

SQL206	Get user's active auctions	R201

```
SELECT product.name, (
    SELECT image.filename
    FROM image
    WHERE image.product_id = product.id
    LIMIT 1
) AS image, product.description, auction.curr_bid, auction.end_date - now()
AS remaining_time
FROM auction
JOIN product ON auction.product_id = product.id
JOIN "user" ON auction.user_id = "user".id
WHERE now() < auction.end_date
AND "user".id = :user_id
```

SQL207	Get user's reviews	R201

```
SELECT review.rating, buyer.id AS reviewer_id, buyer.username AS
reviewer_username, review.date, product.name AS product_name,
review.message, auction.id AS auction_id,
    (SELECT image.filename
    FROM image
    WHERE product.id = image.product_id
    LIMIT 1) AS image_filename
FROM review
INNER JOIN bid ON review.bid_id = bid.id
INNER JOIN auction ON bid.auction_id = auction.id
INNER JOIN "user" buyer ON bid.user_id = buyer.id
INNER JOIN product ON auction.product_id = product.id
INNER JOIN "user" seller ON auction.user_id = seller.id
WHERE seller.id = :user_id
```

SQL208	Get user's wins	R201

```
SELECT product.name AS product_name, product.description, auction.id AS
auction_id, auction.start_bid, auction.curr_bid, auction.end_date,
seller.username AS seller_username, seller.id AS seller_id, bid.id AS
bid_id,
    (SELECT image.filename
    FROM image
    WHERE product.id = image.product_id
    LIMIT 1) AS image_filename
FROM auction
JOIN product ON auction.product_id = product.id
JOIN bid ON auction.id = bid.auction_id
AND auction.curr_bid = bid.amount
JOIN "user" winner ON bid.user_id = winner.id
JOIN "user" seller ON auction.user_id = seller.id
WHERE now() > auction.end_date
AND winner.id = :user_id
```

SQL209	Get following users	R201

```
SELECT us1.id, us1.profile_pic, us1.name, us1.username
FROM follow
INNER JOIN "user" us1 ON follow.user_followed_id = us1.id
INNER JOIN "user" us2 ON follow.user_following_id = us2.id
WHERE us2.id = :following_user_id
```

SQL210	Get last 2 user's reviews	R201

```
SELECT review.id AS review_id, auction.id AS auction_id, seller.username AS
seller_username, seller.id AS seller_id, review.date
FROM review
JOIN bid ON review.bid_id = bid.id
JOIN auction ON bid.auction_id = auction.id
JOIN "user" seller ON auction.user_id = seller.id
JOIN "user" own ON bid.user_id = own.id
WHERE own.id = :user_id
ORDER BY review.date DESC
LIMIT 2
```

SQL211	Get last 2 user's bids	R201

```
SELECT bid.amount, auction.id AS auction_id, bid.date, seller.id AS
seller_id, seller.username AS seller_username
FROM bid
INNER JOIN "user" own ON bid.user_id = own.id
JOIN auction ON bid.auction_id = auction.id
JOIN "user" seller ON auction.user_id = seller.id
WHERE own.id = :user_id
ORDER BY bid.date
LIMIT 2
```

SQL212	Get the last 2 followed users	R201

```
SELECT followed.username AS followed_username, followed.id AS followed_id,
follow.date
FROM follow
JOIN "user" followed ON follow.user_followed_id = followed.id
JOIN "user" following ON follow.user_following_id = following.id
WHERE following.id = :user_id
ORDER BY follow.date
LIMIT 2
```

SQL213	Get the last 2 wins	R201

```
SELECT auction.end_date, seller.username AS seller_username, auction.id AS
auction_id, seller.id AS seller_id
FROM auction
JOIN bid ON auction.id = bid.auction_id AND auction.curr_bid = bid.amount
JOIN "user" winner ON bid.user_id = winner.id
JOIN "user" seller ON auction.user_id = seller.id
WHERE now() > auction.end_date
AND winner.id = :user_id
ORDER BY auction.end_date DESC
LIMIT 2
```

SQL214	Get the last 2 question posted by the user	R201

```
SELECT own.username AS own_username, question.id AS question_id, auction.id
AS auction_id, seller.username AS seller_username, question.date
FROM question
JOIN "user" own ON question.user_id = own.id
JOIN auction ON question.auction_id = auction.id
JOIN "user" seller ON auction.user_id = seller.id
WHERE own.id = :user_id
ORDER BY question.date DESC
LIMIT 2
```

SQL215	Get the last 2 added auctions on the watchlist	R201

```
SELECT "user".username, auction.id, product.name, watchlist.date
FROM watchlist
JOIN "user" ON watchlist.user_id = "user".id
JOIN auction ON watchlist.auction_id = auction.id
JOIN product ON auction.product_id = product.id
WHERE "user".id = :user_id
ORDER BY watchlist.date DESC
LIMIT 2
```

SQL216	Insert review	R210

```
INSERT INTO review (rating, message, DATE, bid_id)
VALUES (:rating, :message, now(), :bid_id)
```

SQL217	Unfollow user	R208

```
DELETE FROM follow
WHERE user_following_id = :following_user_id
AND user_followed_id = :followed_user_id
```

SQL218	Update user's details	R204

```
UPDATE "user"
SET name = :real_name,
short_bio = :small_bio,
email = :email,
phone = :phone,
full_bio = :full_bio
WHERE id = :user_id
```

1.3 M03: Auction

SQL301	What are the last 5 bids of an auction?	R301

```
SELECT "user".username, bid.amount, bid.date
FROM auction
INNER JOIN bid ON auction.id = bid.auction_id
INNER JOIN "user" ON bid.user_id = "user".id
WHERE auction.id = ?
ORDER BY bid.date DESC
LIMIT 5;
```

SQL302	How many bids were made in a certain auction?	R301

```
SELECT COUNT(*)
FROM bid
JOIN auction ON bid.auction_id = auction.id
WHERE auction.id = ?;
```

SQL303	How many reviews does the seller has?	R301

```
SELECT COUNT(*)
FROM review
INNER JOIN bid ON review.bid_id = bid.id
INNER JOIN auction ON bid.auction_id = auction.id
INNER JOIN "user" ON auction.user_id = "user".id
WHERE "user".id = ?;
```

SQL304	Get similar auctions to a given auction	R301

```
SELECT similarAuction.id, similarProduct.name, (
SELECT image.filename
FROM image
WHERE similarProduct.id = image.product_id
LIMIT 1
) AS image
FROM auction originalAuction
JOIN auction similarAuction ON originalAuction.id != similarAuction.id
JOIN product originalProduct ON originalAuction.product_id =
originalProduct.id
JOIN product similarProduct ON similarAuction.product_id = similarProduct.id
WHERE similarAuction.type = originalAuction.type
AND similarProduct.type && originalProduct.type
AND originalAuction.id = ?
LIMIT 8;
```

SQL305	Bid on the auction	R311

```
INSERT INTO bid (amount, DATE, user_id, auction_id)
 VALUES (:amount_bid, now(), :user_id, :auction_id);
```

SQL306	Get auction questions	R301

```
SELECT question.*, "user".username
FROM question
  INNER JOIN "user" ON "user".id = question.user_id
WHERE auction_id = ?;
```

SQL307	Get the answer to a given question	R301

```
SELECT * FROM answer WHERE question_id = ?;
```

SQL308	Create a question	R306

```
INSERT INTO question (DATE, message, auction_id, user_id) VALUES (now(), ?,
?, ?);
```

SQL309	Create an answer	R307

```
INSERT INTO answer (DATE, message, question_id, user_id) VALUES (now(), ?,
?, ?);
```

SQL310	Create auction	R304

```
INSERT INTO auction (start_bid, curr_bid, start_date, end_date, TYPE,
user_id, product_id, DATE) VALUES (?, ?, ?, ?, ?, ?, ?, now());
```

SQL311	Add images to an auction	R305

```
INSERT INTO image (filename, product_id, description) VALUES (?, ?, ?)
```

1.4 M04: Auctions and Watchlist

SQL401	Get all categories.	R402, R401, R504

```
SELECT unnest(enum_range(NULL::category_type))::text;
```

SQL402	Get number of active auctions.	R402

```
SELECT COUNT(*) FROM auction WHERE now() < end_date;
```

SQL403	Get total value of active auctions.	R402

```
SELECT SUM(curr_bid) FROM auction WHERE now() < end_date;
```

SQL404	Get top ten ranking users.	R402

```
SELECT "user".id, "user".username, "user".rating
FROM "user"
WHERE "user".rating IS NOT NULL
ORDER BY rating DESC
LIMIT 10;
```

SQL405	Get most popular auctions.	R402

```
SELECT auction.id, product.name AS product_name, "user".username,
"user".rating AS user_rating,
auction.curr_bid, auction.end_date, "user".id AS user_id
FROM bid
INNER JOIN auction ON bid.auction_id = auction.id
INNER JOIN product ON auction.product_id = product.id
INNER JOIN "user" ON auction.user_id = "user".id
WHERE now() < auction.end_date
GROUP BY auction.id, product.name, "user".username, "user".rating,
auction.curr_bid, auction.end_date, "user".id
ORDER BY COUNT(*) DESC
LIMIT 15;
```

SQL406	Search auctions by name.	R401

```
SELECT auction.id, product.name AS product_name, product.description,
"user".username,
"user".rating AS user_rating, auction.curr_bid, auction.end_date,
"user".id AS user_id, ts_rank_cd(textsearch, query) AS rank
FROM auction, product, "user",
plainto_tsquery('english', :textSearch) AS query,
to_tsvector('english', product.name || ' ' || product.description) AS
textsearch
WHERE auction.product_id = product.id AND query @@ textsearch
AND now() < auction.end_date
AND auction.user_id = "user".id
ORDER BY rank DESC;
```

SQL407	Search auctions by date, price and category.	R404

```
SELECT auction.id, product.name AS product_name, product.description,
"user".username, "user".rating AS user_rating, auction.curr_bid,
auction.end_date, "user".id AS user_id
FROM auction, product, "user"
WHERE auction.product_id = product.id AND auction.user_id = "user".id AND
:fromDate < auction.end_date AND auction.end_date < :toDate AND
auction.curr_bid >= :fromPrice AND auction.curr_bid <= :toPrice AND
:category = ANY(product.type);
```

SQL408	Get the auctions in the watchlist of a certain user.	R403

```
SELECT product.name,
(SELECT image.filename
FROM image
WHERE image.product_id = product.id
LIMIT 1) AS image,
product.description, auction.curr_bid, now() - auction.end_date AS
remaining_time,
seller.username AS seller_username, seller.rating
FROM watchlist
JOIN auction ON watchlist.auction_id = auction.id
JOIN product ON auction.product_id = product.id
JOIN "user" own ON watchlist.user_id = own.id
JOIN "user" seller ON auction.user_id = seller.id
WHERE own.id = ?;
```

SQL409	Add auction to watchlist.	R405

```
INSERT INTO watchlist (auction_id, user_id, notifications, DATE)
VALUES (?, ?, ?, ?);
```

SQL410	Toggle notification option of an auction in the user's watchlist.	R406

```
UPDATE watchlist
SET notifications = NOT notifications
WHERE auction_id = ?
AND user_id = ?;
```

SQL411	Remove auction from watchlist.	R407

```
DELETE FROM watchlist
WHERE auction_id = ?
AND user_id = ?;
```

SQL412	What is the main image of an auction?	R401, R402

```
SELECT image.*
FROM image
JOIN product ON image.product_id = product.id
JOIN auction ON product.id = auction.product_id
WHERE auction.id = ?
LIMIT 1;
```

1.5 M05: Administrators

SQL501	Adds a new admin to the database	R506

```
INSERT INTO admin (username, hashed_pass, email) VALUES (?, ?, ?)
```

SQL502	Adds a new product category	R508

```
ALTER TYPE category_type ADD VALUE :title
```

SQL503	Deletes an auction	R511

```
DELETE FROM auction WHERE id = ?
```

SQL504	Deletes an user	R510

```
DELETE FROM "user" WHERE id = ?
```

SQL505	Get all users	R502

```
SELECT * FROM "user" ORDER BY id ASC;
```

SQL506	Get all auctions	R503

```
SELECT * FROM auction ORDER BY id ASC;
```

SQL507	Retrieves all the feedback given by the users

```
SELECT * FROM feeback ORDER BY DATE DESC;
```

SQL508	Creates a new notification associated to an user	R507
`INSERT INTO notification (message, TYPE, is_new, user_id, DATE) VALUES (?, ?, ?, ?, now()))`		

2 Transactions

T01	Bid transaction
Isolation Level	REPEATABLE READ, because only sees data committed before the transaction began, ensuring that the balance of the user is equal to the value initially read at the beginning of the INSERT and UPDATE statements. In fact, the first SELECT locks the *amount* attribute of table *user* until the end of the transaction.
Web Resource	R311

```
BEGIN TRANSACTION ;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;

SELECT amount AS balance FROM "user" WHERE "user".id = :user_id;

-- if balance >= amount_bid

    INSERT INTO bid (amount, DATE, user_id, auction_id)
    VALUES (:amount_bid, now(), :user_id, :auction_id);

    UPDATE "user"
    SET amount = amount - :amount_bid
    WHERE "user".id = :user_id;
    -- send success message

-- else
    -- send error message

COMMIT;
```

T02	Get information of auctions in homepage
Isolation Level	SERIALIZABLE READ ONLY, because in the middle of the transaction the insertion of new rows in the *auction* table can occur, which will lead to incompatibilities between the SELECTS, resulting in a *Phantom Read* (If a new auction is entered between the 1st SELECT and the 2nd SELECT, the 2nd SELECT will read more auctions than the first one). It is READ ONLY because only use SELECT instructions.
Web Resource	R402

```
BEGIN TRANSACTION ;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY ;

-- get total active auctions
SELECT COUNT(*)
FROM auction
WHERE now() < end_date;

-- get total value of all auctions
SELECT SUM(curr_bid)
FROM auction
WHERE now() < end_date;

-- get most popular auctions of site (more popular = more bids)
SELECT auction.id, product.name AS product_name, "user".username,
"user".rating AS user_rating,
auction.curr_bid, auction.end_date, "user".id AS user_id
FROM bid
INNER JOIN auction ON bid.auction_id = auction.id
INNER JOIN product ON auction.product_id = product.id
INNER JOIN "user" ON auction.user_id = "user".id
WHERE now() < auction.end_date
GROUP BY auction.id, product.name, "user".username, "user".rating,
auction.curr_bid, auction.end_date, "user".id
ORDER BY COUNT(*) DESC
LIMIT 15;

COMMIT;
```

T03	Get auction main information
Isolation Level	REPEATABLE READ READ ONLY, because, in the middle of the transaction, the update of the specified auction row can occur, which will lead to incompatibilities between the SELECTS, resulting in a *Nonrepeatable Read*. It is READ ONLY because it only uses SELECT instructions.
Web Resource	R301

```
BEGIN TRANSACTION ;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ ONLY ;

SELECT *
FROM auction
WHERE auction.id = ?;

SELECT product.*
FROM product
JOIN auction ON auction.product_id = product.id
WHERE auction.id = ?;

SELECT "user".*
FROM "user"
INNER JOIN auction ON "user".id = auction.user_id
WHERE auction.id = ?;

SELECT image.*
FROM image
JOIN product ON image.product_id = product.id
JOIN auction ON product.id = auction.product_id
WHERE auction.id = ?;

COMMIT;
```

T04	Post answer and send notification to user
Isolation Level	REPEATABLE READ, because it prevents the update of notifications options in the middle of transaction.
Web Resource	R307

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert answer.
INSERT INTO answer(DATE, message, question_id, user_id, auction_id)
VALUES(now(), :message, :question_id, :user_id, :auction_id);

-- Get question user id.
SELECT user_id
FROM question
WHERE id = :question_id;

-- Get user notification enabled/disabled setting.
SELECT notifications
FROM watchlist
WHERE user_id = :user_id
AND auction_id = :auction_id;

-- if notifications are enabled

   -- Insert notification.
   INSERT INTO notification(message, TYPE, user_id, is_new, DATE)
   VALUES('Your question at the auction was answered!',
      'Answer',
      :user_id,
      TRUE,
      now());

   -- send success message with notification.

-- else

   --send success message without notification.

COMMIT;
```

T05	Get questions and respective answers, if they exist
Isolation Level	REPEATABLE READ READ ONLY, because, in the middle of the transaction, the update of retrieved data can occur, which will lead to incompatibilities between the SELECTS, resulting in a *Nonrepeatable Read*. It is READ ONLY because it only uses SELECT instructions.
Web Resource	R301

```
BEGIN TRANSACTION ;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ ONLY ;

SELECT question.*, "user".username
FROM question
  INNER JOIN "user" ON "user".id = question.user_id
WHERE auction_id = ?;

-- foreach question
SELECT * FROM answer WHERE question_id = ?;

COMMIT;
```