A9: Main accesses to the database and transactions

Our project features an information system capable of supporting an online store, which would allow users to buy products from a wide range of categories. In this artefact we present its main accesses to the database, including transactions.

1 Main accesses to the database

Main accesses to the database.

1.1 M01: Authentication and Individual Profile

SQL101	Creates a new user in the platform
Web Resource	R105

```
INSERT INTO users(id, firstName, lastName, username, email, password, imageURL,
        dateCreated, dateModified, active, remember_token)
VALUES (DEFAULT, $firstName, $lastName, $username, $email, $password, $imageURL,
        DEFAULT, DEFAULT, true, $token);
```

Table 1: Authentication and Individual Profile

1.2 M02: View Products

SQL102	Shows the list of all products
Web Resource	R201

```
SELECT p.id AS "ID",p.name AS "Name", p.quantityInStock AS "Quantity In Stock",
           p.dateCreated AS "Date Created",
        p.price AS "Price",
           p.imageURL AS "Image URL", p.bigDescription AS "Big Description",
       p.shortDescription AS "Short Description",
           (array_agg(pc.categoryName ORDER BY p.id DESC))[1] AS "Category",
       (array_agg(b.brandname ORDER BY p.id DESC))[1] AS "Brand",
       AVG(pr.rating) AS "Rating"
FROM products p, categories pc, brands b, reviews pr
WHERE p.id_brand = b.id_brand AND p.id_category = pc.id_category AND pr.id_product = p.id
GROUP BY p.id;
```

Table 2: View Products

1.3 M03: View Product

SQL103	Shows a particular product
Web Resource	R202

```
SELECT p.id AS "ID",p.name AS "Name", p.quantityInStock AS "Quantity In Stock",
           p.dateCreated AS "Date Created",
        p.price AS "Price",
           p.imageURL AS "Image URL", p.bigDescription AS "Big Description",
       p.shortDescription AS "Short Description",
           (array_agg(pc.categoryName ORDER BY p.id DESC))[1] AS "Category",
       (array_agg(b.brandname ORDER BY p.id DESC))[1] AS "Brand",
       AVG(pr.rating) AS "Rating"
FROM products p, categories pc, brands b, reviews pr
WHERE p.id_brand = b.id_brand AND p.id_category = pc.id_category AND pr.id_product = p.id
AND p.id = $productID
GROUP BY p.id;
```

Table 3: View Product

1.4 M04: Search Products

SQL104	Searches products by name and category
Web Resource	R207

```
SELECT p.id AS "ID",p.name AS "Name", p.quantityInStock AS "Quantity In Stock",
           p.dateCreated AS "Date Created", p.price AS "Price",
           p.imageURL AS "Image URL", p.bigDescription AS "Big Description",
       p.shortDescription AS "Short Description",
           (array_agg(pc.categoryName ORDER BY p.id DESC))[1] AS "Category",
       (array_agg(b.brandname ORDER BY p.id DESC))[1] AS "Brand",
       AVG(pr.rating) AS "Rating", ts_rank_cd(document, query) AS rank
FROM products p, categories pc, brands b, reviews pr,
to_tsvector(pc.categoryName || ' ' || p.name) AS document,
plainto_tsquery($query) AS query
WHERE p.id_brand = b.id_brand AND p.id_category = pc.id_category
AND pr.id_product = p.id AND document  @@ query
GROUP BY p.id, document.document, query.query
ORDER BY  rank DESC;
```

Table 4: Search Products

1.5 M05: Consult Wishlist

SQL105	Shows the products in the user wishlist
Web Resource	R303

```
SELECT products.name, products.price, products.imageURL, clients.id_client
FROM products, clients, wishlists
        WHERE clients.id_client =wishlists.id_client
        AND products.id = wishlists.id_product
        AND clients.id_client = $clientID;
```

Table 5: Consult Wishlist

1.6 M06: Add Product to Wishlist

SQL106	Adds a product to the wishlist
Web Resource	R304

```
INSERT INTO productwishlist (id_product, id_client)
VALUES ($id_product, $id_client);
```

Table 6: Add Product to Wishlist

1.7 M07: Consult Cart

SQL107	Consults the list of products in cart
Web Resource	R402

```
SELECT products.name, products.price,
products.imageURL, clients.id_client
FROM products, clients, carts
        WHERE clients.id_client = carts.id_client
        AND products.id = carts.id_product
        AND clients.id_client = $clientId;
```

Table 7: Consult Cart

1.8 M08: Add to Cart

SQL108	Adds product to cart
Web Resource	R402

```
INSERT INTO carts (id_client, id_product, quantity)
VALUES ($id_client, $id_product, $quantity);
```

Table 8: Add to Cart

1.9 M09: Delete from Cart

SQL109	Deletes a product from cart
Web Resource	R403

```
DELETE FROM carts WHERE id_product=$id_product AND id_client=$id_client;
```

Table 9: Delete from Cart

1.10 M10: Consult purchased Products

SQL110	Retrieve the products bought by an user
Web Resource	R404

```
SELECT products.name, products.price, products.imageURL,
purchaseproducts.quantity, purchaseproducts.cost
FROM purchases, products, purchaseproducts
        WHERE purchases.id = purchaseproducts.id_purchase
        AND products.id = purchaseproducts.id_product
        AND purchases.id_client = $id_client;
```

Table 10: Consult purchased Products

1.11 M11: Make a Purchase

SQL111	Retrieve the products bought by an user
Web Resource	R405

```
INSERT INTO purchases(id, id_client, id_address, purchaseDate, purchaseState,
        cost, paymentType, cardNumber, cardName, cardExpirationDate, nif)
VALUES ($id, $id_client,$id_address, DEFAULT, $purchaseState,
        $cost, $paymentType, $cardNumber, $cardName, $cardExpirationDate, $nif);
```

Table 11: Make a Purchase

2 Transactions

Transactions needed to ensure the integrity of the data, with a proper justification.

T01	Retrieve the list of all the products available in our platform
Isolation Level	SERIALIZABLE READ ONLY
Justification	In order for the information retrieved in both SELECTS to be the same we must assure that no new rows can be inserted in the table *products*, that is, it must be assured that no *Phantom Read's* can occur. That's why the isolation level of this transaction must be SERIALIZABLE. It is also READ ONLY as only SELECTS are used.

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

--Get number of purchases
SELECT COUNT(*)
FROM products ;

--Get products
SELECT p.id AS "ID",p.name AS "Name", p.quantityInStock AS "Quantity In Stock",
           p.dateCreated AS "Date Created",
        p.price AS "Price",
           p.imageURL AS "Image URL", p.bigDescription AS "Big Description",
       p.shortDescription AS "Short Description",
           (array_agg(pc.categoryName ORDER BY p.id DESC))[1] AS "Category",
       (array_agg(b.brandname ORDER BY p.id DESC))[1] AS "Brand",
       AVG(pr.rating) AS "Rating"
FROM products p, categories pc, brands b, reviews pr
WHERE p.id_brand = b.id_brand AND p.id_category = pc.id_category AND pr.id_product = p.id
GROUP BY p.id;

COMMIT;
```

Table 12: Products

T02	Inserts a new client
Isolation Level	REPEATABLE READ
Justification	In order to maintain consistency both INSERTS need to be executed. If an error occurs, a ROLLBACK is issued. The isolation level is REPEATABLE READ as it must be assured that an update of the attribute *id* in table *users* does not occur in at the same time of this transaction, because it would trample the data.

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

DO $$
DECLARE idTemp integer;
BEGIN
  INSERT INTO users(firstName, lastName, username, email, password, imageURL,
        dateCreated, dateModified, active)
VALUES ($firstName, $lastName, $username, $email, $password, $imageURL, DEFAULT, DEFAULT,
true) RETURNING id INTO idTemp ;

   INSERT INTO clients VALUES(idTemp, $cellPhone);
END $$;

COMMIT;
```

Table 13: Client

T03	Inserts a new Chat Support
Isolation Level	REPEATABLE READ
Justification	In order to maintain consistency both INSERTS need to be executed. If an error occurs, a ROLLBACK is issued. The isolation level is REPEATABLE READ as it must be assured that an update of the attribute *id* in table *users* does not occur in at the same time of this transaction, because it would trample the data.

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

DO $$
DECLARE idTemp integer;
BEGIN
  INSERT INTO users(firstName, lastName, username, email, password, imageURL,
        dateCreated, dateModified, active)
VALUES ($firstName, $lastName, $username, $email, $password, $imageURL, DEFAULT, DEFAULT,
true) RETURNING id INTO idTemp ;

  INSERT INTO chatSupports VALUES(idTemp);
END $$;

COMMIT;
```

Table 14: Chat Support

T04	Inserts a new Brand Manager
Isolation Level	REPEATABLE READ
Justification	In order to maintain consistency both INSERTS need to be executed. If an error occurs, a ROLLBACK is issued. The isolation level is REPEATABLE READ as it must be assured that an update of the attribute *id* in table *users* does not occur in at the same time of this transaction, because it would trample the data.

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

DO $$
DECLARE idTemp integer;
BEGIN
  INSERT INTO users(firstName, lastName, username, email, password, imageURL,
        dateCreated, dateModified, active)
VALUES ($firstName, $lastName, $username, $email, $password, $imageURL, DEFAULT, DEFAULT,
true) RETURNING id INTO idTemp ;

  INSERT INTO brandManagers VALUES(idTemp);
END $$;

COMMIT;
```

Table 15: Brand Manager

T05	Inserts a new Admin
Isolation Level	REPEATABLE READ
Justification	In order to maintain consistency both INSERTS need to be executed. If an error occurs, a ROLLBACK is issued. The isolation level is REPEATABLE READ as it must be assured that an update of the attribute *id* in table *users* does not occur in at the same time of this transaction, because it would trample the data.

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

DO $$
DECLARE idTemp integer;
BEGIN
  INSERT INTO users(firstName, lastName, username, email, password, imageURL,
        dateCreated, dateModified, active)
VALUES ($firstName, $lastName, $username, $email, $password, $imageURL, DEFAULT, DEFAULT,
true) RETURNING id INTO idTemp ;

  INSERT INTO brandManagers VALUES(idTemp);

  INSERT INTO admins VALUES(idTemp);
END $$;

COMMIT;
```

Table 16: Admin

T06	Buy a product
Isolation Level	REPEATABLE READ
Justification	In order to maintain consistency both INSERTS need to be executed. If an error occurs, a ROLLBACK is issued. The isolation level is REPEATABLE READ as it must be assured that an update of the attribute *id* in table *purchases* does not occur in at the same time of this transaction, because it could trample the data.

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

DO $$
DECLARE idp integer;
BEGIN
INSERT INTO purchases(id_client, id_address, purchaseDate, purchaseState,
        cost, paymentType, cardNumber, cardName, cardExpirationDate, nif)
VALUES ($id_client,$id_address, DEFAULT, $purchaseState,
        0, $paymentType, $cardNumber, $cardName, $cardExpirationDate, $nif)
    RETURNING id_purchase INTO idp ;

INSERT INTO purchaseproducts (id_purchase, id_product, quantity, cost)
SELECT idp, id_product, quantity, 0
FROM carts
WHERE carts.id_client = $id_client;

DELETE FROM carts WHERE id_client = $id_client;

END $$;

COMMIT;
```

Table 17: Purchase

Revision history
Changes made to the first submission:

- The following queries were added to the list of accesses: SQL102, SQL103, SQL105, SQL106, SQL107, SQL109
- Changed Transaction T01. Instead of showing the product bought by an user, now it shows the products available in our platform.
- Altered Transaction T02 that adds a new user to the platform. Now this transaction adds the id of the user by default, instead of receiving it as a parameter.
- Added Transactions T03, T04, T05 that insert new Chat Supports's, Brand Manager's and Admin's.
- Added Index04, making this index an example of a full-text search index
- Updated SQL Code

GROUP1736, 21/04/2018

- Group member 1 Beatriz de Henriques Martins, up201502858@fe.up.pt
- Group member 2 Francisco Tuna Andrade, up201503481@fe.up.pt
- Group member 3 Luís Miguel Santos Monteiro Saraiva, up201404302@fe.up.pt
- Group member 4 Ricardo Filipe Amaro Saleiro Abreu, up201304450@fe.up.pt