Code Home

A9: Main Accesses to the database and transactions

This artefact shows the main accesses to the database, including the transactions.

For each transaction, the isolation level is explicitly stated and read-only transactions are identified to improve global performance. For each identified access, the SQL code and the reference of web resources (A7) are provided.

1. Main Accesses

Main accesses to the database.

1.1. M01: Authentication and Individual Profile

SQL101	Login
Web Resource	R101

```
SELECT password_token, auth_type

  FROM "User"

  WHERE "User".username = $username;
```

SQL102	Register
Web Resource	R105

```
INSERT INTO "User"

(id, username, type, pass_token, auth_type, email, state, description, img_path)

VALUES($id, $username, $type, $pass_token,$auth_type, $email, $state, $description, $img_path);
```

SQL103	View HomePage
Web Resource	R107

```
SELECT "User".username, "User".img_path, title, content, points,
"Post".isVisible

  FROM "Question" INNER JOIN "Post" ON "Question".postID = "Post".id
INNER JOIN "User" ON "Post".posterID = "User".id

  WHERE isVisible = TRUE

ORDER BY "Post".date

LIMIT 50;
```

SQL104	View Profile
Web Recource	R108

```
SELECT username, email, description, img_path, points

  FROM "User"

  WHERE "user".id = $userId;
```

SQL105	Edits Profile
Web Resource	R110

```
UPDATE "User" SET email = $email, state = $state, description =

$description, img_path = $img_path,points = $points WHERE id=$id;
```

SQL106	Delete Profile
Web Resource	R111

```
UPDATE "User" SET state = ‘INACTIVE’ WHERE id=$id;
```

1.2. M02: Questions and Answers

SQL201	View Question and Answers to that same question
Web Resource	R201

```
SELECT "User".username, "User".img_path, "Post".content, "Post".points,
"Post".isVisible

 FROM "Post" INNER JOIN "User" ON "Post".posterID = "User".id INNER JOIN
"Question" ON "Question".postID = "Post".id

  WHERE isVisible = TRUE AND "Post".id = $postId





SELECT "User".username, "User".img_path, "Post".content, "Post".points,
"Post".isVisible

  FROM "Post" INNER JOIN "User" ON "Post".posterID = "User".id INNER JOIN
"Answer" ON "Answer".questionID = "Post".id

  WHERE isVisible = TRUE AND "Post".id = $postId

ORDER BY "Post".points

LIMIT 50;
```

SQL202	Vote on post
Web Resource	R202

```
INSERT INTO PostVote VALUES($post_id,  $poster_id, $value);
```

SQL203	Post new Answer
Web Resource	R203

```
BEGIN TRANSACTION;

    -- Insert Associated Post

INSERT INTO Post(id,posterID,content) VALUES($id,$posterID,$content);

    -- Insert Answer

INSERT INTO Answer(postID,questionId) VALUES($id,$questionId);

COMMIT;
```

SQL204	Post new Question
Web Resource	R205

```
BEGIN TRANSACTION;

-- Insert Associated Post

INSERT INTO Post(id,posterID,content) VALUES($id,$posterID,$content);

-- Insert Question

INSERT INTO Question(postID,title) VALUES($id,$title);

--Insert Associated Tag

INSERT INTO TagQuestion(question_Id,tag_id) SELECT $id,tag.id FROM Tag
        tag WHERE tag.name = $tag;

COMMIT;
```

SQL205	Search Question
Web Resource	R206

```
SELECT "User".username, "User".img_path, title, content, points,
"Post".isVisible

  FROM "Question" INNER JOIN "Post" ON "Question".postID = "Post".id
INNER JOIN "User" ON "Post".posterID = "User".id

  WHERE ("Question".title = LIKE %$searchName% OR content LIKE %$
searchName%) And isVisible = TRUE

ORDER BY points

LIMIT 50;
```

SQL206	Report Post
Web Resource	R207

```
INSERT INTO PostReport(postID,reporterID,reason)
VALUES($postID,$reporterID,$reason);
```

SQL207	Delete Question
Web Resource	R208

```
UPDATE Post SET isVisible=FALSE WHERE id=$id;
```

1.3. M03: User Administration and Static pages

SQL301	Ban User
Web Resource	R301

```
BEGIN TRANSACTION;

--Update User Status

 UPDATE users SET status = ‘BANNED’ WHERE id=$id;

-- Insert Ban

 INSERT INTO BanInfo(isPermanent,initDate,endDate,userID,adminID)
VALUES($isPermanent, $initDate, $endDate, $id, $adminID);



COMMIT;
```

SQL302	Unban User
Web Resource	R302

```
BEGIN TRANSACTION;

--Delete BanInfo

 DELETE FROM BanInfo WHERE id=$ban_id;

--Update User Status

 UPDATE users SET status = ‘ACTIVE’ WHERE id=$id;



COMMIT;
```

SQL303	View User Information
Web Recource	R303

```
SELECT username, email, description, img_path, points

  FROM "User"

  WHERE "user".id = $userId;
```

SQL304	Edit User information
Web Resource	R304

```
UPDATE "User" SET email = $email, state = $state, description =

$description, img_path = $img_path, points = $points WHERE id=$id;
```

SQL305	Search User
Web Recource	R307

```
SELECT "User".username, "User".img_path, title, content

  FROM "User"

  WHERE "User".username = LIKE %$searchName%

ORDER BY points

LIMIT 50;
```

SQL306	Remove Post
Web Recource	R308

```
UPDATE Post SET isVisible=FALSE WHERE id=$id;
```

SQL307	View Post Reports
Web Recource	R309

```
 SELECT postId,reporterID,date,reason FROM PostReport WHERE postId=$id
ORDER BY date DESC;
```

SQL308	Close Question
Web Recource	R310

```
  UPDATE Question Set isClose=TRUE WHERE postID=$id;
```

SQL309	Mark Answer As Correct
Web Recource	R311

```
  UPDATE Answer Set isCorrect=TRUE WHERE postID=$id;
```

SQL310	Edit Answer
Web Resource	R313

```
UPDATE Post SET content=$content, date=$date WHERE id=$id;
```

SQL311	Edit question
Web Resource	R315

```
BEGIN TRANSACTION;

-- Update Associated Post

UPDATE "Post" SET content = $content, date = $date, isVisible =
$isVisible, points = $points WHERE id = $id;

-- Update Question

UPDATE "Question" SET isClosed = $isClosed, nViews = $nViews, title =
$title WHERE postID = $postID;

--Insert Associated Tag

UPDATE "TagQuestion" SET tag_id = $tag_id WHERE question_id =
$question_id;

COMMIT;
```

SQL312	View Contacts List
Web Resource	R316

```
SELECT * FROM users INNER JOIN (SELECT name as subjectName, message,

date, userID FROM Contact INNER JOIN Subject ON

Contact.subjectID=Subject.subjectID) AS contact ON

users.id=contact.userID ORDER BY date DESC;
```

2. Transactions

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

T01	Add Question
Isolation level	READ COMMITED
Justification	Since we are adding a row to different tables where there might be concurrent insertions, we need to keep data consistency, and in case an error occurs during the insertion we need to roll back the whole block as a question cannot exist without an associated post and there can be no associated tags to a nonexistent question. We also obtain the tag id by reading from its table therefore we want this data to be consistent.

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL READ COMMITED



-- Insert Associated Post

INSERT INTO Post(id,posterID,content) VALUES($id,$posterID,$content);



-- Insert Question

INSERT INTO Question(postID,title) VALUES($id,$title);



--Insert Associated Tag

INSERT INTO TagQuestion(question_Id,tag_id) SELECT $id, tag.id FROM Tag
tag WHERE tag.name = $tag;



COMMIT;
```

T02	Add Answer
Isolation level	READ COMMITED
Justification	Since we are adding a row to different tables where there might be concurrent insertions we need to keep data consistency, and in case an error occurs during the insertion we need to roll back the whole block as an answer cannot exist without an associated post.

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL READ COMMITED



-- Insert Associated Post

INSERT INTO Post(id,posterID,content) VALUES($id,$posterID,$content);



-- Insert Answer

INSERT INTO Answer(postID,questionId) VALUES($id,$questionId);



COMMIT;
```

T03	Ban User
Isolation level	SERIALIZABLE READ WRITE
Justification	To maintain consistency, it's necessary to use a transaction to ensure that all the code executes without errors. If an error occurs, a ROLLBACK is issued. We also need to make sure that a user is not banned and then unbanned due to concurrent behavior and as such we use serializable isolation level.

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE



--Update User Status

 UPDATE users SET status = ‘BANNED’ WHERE id=$id;



-- Insert Ban

 INSERT INTO BanInfo(isPermanent,initDate,endDate,userID,adminID)
VALUES($isPermanent, $initDate, $endDate, $id, $adminID);



COMMIT;
```

T04	UnBan User
Isolation level	SERIALIZABLE READ WRITE
Justification	To maintain consistency, it's necessary to use a transaction to ensure that all the code executes without errors. If an error occurs, a ROLLBACK is issued. We also need to make sure that a user is not banned and then unbanned due to concurrent behavior and as such we use serializable isolation level.

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE



--Update User Status

 UPDATE users SET status = ‘ACTIVE’ WHERE id=$id;



 --Delete BanInfo

 DELETE FROM BanInfo WHERE id=$ban_id;



COMMIT;
```

Group

- Davide Henrique Fernandes da Costa, up201503995@fe.up.pt
- Dinis Filipe da Silva Trigo, up201504196@fe.up.pt
- Diogo Afonso Duarte Reis, up201505472@fe.up.pt
- Tiago José Sousa Magalhães, up201607931@fe.up.pt