A9: Main accesses to the database and transactions

This artefact shows the main accesses to the database, as well as the transactions necessary for ensuring database consistency.

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

This section describes the main accesses to the database.

SQL101	Search all questions by their title
Web Resource	R301

```
SELECT *
FROM questions
WHERE
  tstitle @@ plainto_tsquery('english', $search_query)
ORDER BY
  ts_rank(tstitle, plainto_tsquery('english', $search_query))
  DESC;
```

> The `tstitle` column is of type “tsvector”, and is equal to `to_tsvector('english', title)`, as advised in class. This column is kept consistent with the title by updating it with triggers “on insert” and “on update”.

SQL102	Get recent questions
Web Resource	R301

```
SELECT questions.id, title, correct_answer,
  score, is_banned, author, content, creation_time
FROM questions, commentables, messages, message_versions
WHERE
  questions.id = commentables.id AND
  commentables.id = messages.id AND
  messages.id = message_versions.message_id AND
  messages.latest_version = message_versions.id
ORDER BY
  message_versions.creation_time DESC
```

```
LIMIT 25;
```

SQL103	Get a question's details
Web Resource	R306

```
SELECT *
FROM questions, messages, message_versions
WHERE
  questions.id = $question_id AND
  questions.id = messages.id AND
  messages.latest_version = message_versions.id;
```

SQL104	Post a questions
Web Resource	R307

```
INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
INSERT INTO commentables (id) VALUES (new_id);
INSERT INTO questions (id, title) VALUES (new_id, $title);
INSERT INTO message_versions (content, message_id) values ($content, new_id);
```

SQL105	Post an answer
Web Resource	R314

```
INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
INSERT INTO commentables (id) VALUES (new_id);
INSERT INTO answers (id, question_id) VALUES (new_id, $question);
INSERT INTO message_versions (content, message_id) values ($content, new_id);
```

SQL106	Post a comment on a question or answer
Web Resource	R318 and R322

```
INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
INSERT INTO comments (id, commentable_id) VALUES (new_id, $commentable);
INSERT INTO message_versions (content, message_id) values ($content, new_id);
```

SQL107	Get a question's answers and their details
Web Resource	R312 and R313

```
SELECT *
FROM messages, answers, message_versions
```

```
WHERE
  answers.question_id = $question_id AND
  messages.id = answers.id AND
  messages.latest_version = message_versions.id
ORDER BY
  message_versions.creation_time
  DESC;
```

SQL108	Get a commentable's comments (question or answer).
Web Resource	R317 and R321

```
SELECT *
FROM commentables, messages, message_versions, comments
WHERE
  commentables.id = $commentable_id AND
  commentables.id = messages.id AND
  comments.commentable_id = commentables.id AND
  message_versions.id = messages.latest_version
ORDER BY
  message_versions.creation_time
  DESC;
```

2. Transactions

Transactions needed to assure the integrity of the data.

T01	Insert a question
Justification	Since a object-oriented approach was used to map the question UML subclass to the relational model, atomicity must be guaranteed so all classes that need to be inserted are in fact all or none inserted.
Isolation level	Read Uncommitted

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

DO $$
DECLARE
  new_id INTEGER;
BEGIN
  INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
  INSERT INTO commentables (id) VALUES (new_id);
  INSERT INTO questions (id, title) VALUES (new_id, $title);
  INSERT INTO message_versions (content, message_id) values ($content, new_id);
END $$;

COMMIT;
```

T02	Insert an answer
Justification	Since a object-oriented approach was used to map the answer UML subclass to the relational model, atomicity must be guaranteed so all classes that need to be inserted are in fact all or none inserted.
Isolation level	Read Uncommitted

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

DO $$
DECLARE
  new_id INTEGER;
BEGIN
  INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
  INSERT INTO commentables (id) VALUES (new_id);
  INSERT INTO answers (id, question_id) VALUES (new_id, $question);
  INSERT INTO message_versions (content, message_id) values ($content, new_id);
END $$;

COMMIT;
```

T03	Insert a comment
Justification	Since a object-oriented approach was used to map the comment UML subclass to the relational model, atomicity must be guaranteed so all classes that need to be inserted are in fact all or none inserted.
Isolation level	Read Uncommitted

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

DO $$
```

```
DECLARE
  new_id INTEGER;
BEGIN
  INSERT INTO messages (author) VALUES ($author) RETURNING id INTO new_id;
  INSERT INTO comments (id, commentable_id) VALUES (new_id, $commentable);
  INSERT INTO message_versions (content, message_id) values ($content, new_id);
END $$;

COMMIT;
```

Several other user stories would require the use of transactions. Fortunately, by default all statements like `insert`, `update` or `delete` are inside their own transaction and the triggers that are fired by them also run in the same transaction. Therefore if anything fails in the execution of the trigger, then everything is rolled back, including the initial statement that fired the trigger.

We've made extensive use of this autocommit mode to ensure database integrity. For example, when a vote is made to a message, a trigger updates its score value. If these two actions weren't executed inside the same transaction and there was a failure in updating the score attribute, then the number of positive votes minus the negative ones wouldn't match the score, leaving the database in a inconsistent state. This is true for all derived attributes, whose consistency is ensured by triggers.

In this example the trigger TRIGGER06 can be replaced by the following transaction:

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

INSER INTO votes (message_id, user_id, positive)
  values ($message_id, $user_id, $positive);
IF $positive THEN
  UPDATE messages
    SET score = score + 1
    WHERE $message_id = messages.id;
ELSIF NOT $positive THEN
  UPDATE messages
    SET score = score - 1
    WHERE $message_id = messages.id;
END IF;

COMMIT;
```

Revision history

- 01/05/2018: Added links to resources.

- 02/05/2018: Added example transaction that would be needed if triggers weren't run in the same transaction as the statement which fired it.

GROUP1763, 24/04/2018

André Cruz, up201503776@fe.up.pt
Daniel Marques, up201503822@fe.up.pt
Edgar Carneiro, up201503784@fe.up.pt
João Carvalho, up201504875@fe.up.pt