A9 – Main Accesses to the database and transactions

This artefact shows the main accesses to the database in the Project Management application. All the main accesses are described with the corresponding SQL code and web resource reference. It also includes all the required transactions for this application.

1. **Main Accesses**

1.1. M01 ~~Create new user~~Registration and Authentication

SQL101	Creates new user in the platform
Web resource	R105
`INSERT INTO User (e_mail, password, URL, username)` `VALUES ($e_mail, $password, $URL, $username)`	

SQL102	Verifies existence of e-mail in the user table
Web resource	R102

`CREATE FUNCTION check_exist_email() RETURNS TRIGGER AS`
`$BODY$`
`BEGIN`
`IF EXISTS (SELECT e_mail FROM "User" WHERE NEW.e_mail = e_mail) THEN`
`RAISE EXCEPTION 'The e-mail you are trying to use is already registred'`
`END IF;`
`RETURN NEW;`
`END`
`$BODY$`
`LANGUAGE plpgsql;`
 	
`CREATE TRIGGER check_exist_email`
`BEFORE INSERT ON "User"`
`FOR EACH ROW`
`EXECUTE PROCEDURE check_exist_user();`	

SQL103	Checks what is the password associated to a certain e-mail in the user table
Web resource	R102
`SELECT password FROM "User"` `WHERE user.e_mail = $userEmail`	

1.2. M02 User Pages~~Projects~~

SQL201	All user's projects
Web resource	R113
SELECT * FROM Project INNER JOIN Project_team ON Project.id=Project_team.id_project WHERE Project.id_coordinator=$user OR Project_team.id_user=$user	

SQL202	All user's personal events
Web resource	R107
SELECT * FROM Personal_event WHERE Personal_event.id_user=$user	

SQL203	All user's board meetings
Web resource	R107
SELECT * FROM Meeting INNER JOIN Board_team ON Meeting.id_board=Board_team.id_board WHERE Board_team.id_worker=$user	

SQL204	User's personal info
Web resource	R106
SELECT * FROM "User" WHERE User.id=$user	

SQL205	Search projects
Web resource	R115
SELECT id, name, description, start_date, ts_rank_cd(textsearch, query) AS rank FROM Project, to_tsquery($search) AS query, to_tsvector(name || ' ' || description) AS textsearch WHERE query @@ textsearch\\ORDER BY rank DESC;	

SQL206	Add personal event
Web resource	*Resource not specified
INSERT INTO Personal_event (date, place, name, id_user) VALUES ($date, $place, $name, $id_user)	

SQL207	Add new project
Web resource	R202
INSERT INTO Project (description, name, id_coordinator, privacy) VALUES ($description, $name, $id_coordinator, $privacy)	

SQL208	Change user's full name
Web resource	R109
UPDATE User SET full_name=$full_name	

SQL209	Change user's password
Web resource	R109
UPDATE User SET password=$password	

SQL210	Change profile picture
Web resource	R109
INSERT INTO Profile_picture (id_user, path) VALUES ($id_user, $path)	

SQL211	Mark notification as read/unread
Web resource	*Resource not specified
UPDATE Notification SET read=$read	

1.3. M03 Project Pages

~~**SQL201**~~	~~Search project~~
~~**Web resource**~~	~~R603~~
~~SELECT id, name, description, start_date, ts_rank_cd(textsearch, query) AS rank FROM Project, to_tsquery($search) AS query, to_tsvector(name || ' ' || description) AS textsearch WHERE query @@ textsearch\\ ORDER BY rank DESC;~~	

SQL301	Project boards
Web resource	R213
SELECT * FROM Board WHERE Board.id_project=$projectId	

SQL302	Project information
Web resource	R203
SELECT * FROM Project WHERE Project.id=$projectId	

SQL303	Project workers and information
Web resource	R205

```
SELECT User.full_name, Profile_picture.path, User.e_mail
FROM "User"
INNER JOIN Profile_picture ON Profile_picture.id_user=$user
WHERE Project.id=$projectId AND
Profile_picture.path=SELECT(MAX(Profile_picture.id))
```

SQL304	Project events
Web resource	R204

```
SELECT *
FROM Meeting
INNER JOIN Board ON Board.id=Meeting.id_board
WHERE Board.id_project=$projectId
```

SQL305	Project forum messages
Web resource	R212

```
SELECT *
FROM Message
WHERE Message.id_project=$projectId AND Message.date >=
DATEADD(day, -7, GETDATE())
```

SQL306	Search users
Web resource	R206

```
SELECT username, full_name, e_mail FROM User
INNER JOIN Project_team ON User.id=Project_team.id_user
WHERE Project_team.id_project=$id_project AND (username LIKE
%'$search%' OR full_name LIKE %'$search%');
ORDER BY username;
```

SQL307	Search boards
Web resource	R214

```
SELECT id, name, description ts_rank_cd(textsearch, query) AS
rank
FROM Board, to_tsquery($search) AS query, to_tsvector(name ||
' ' || description) AS textsearch
WHERE query @@ textsearch\\ AND id_project=$id_project
ORDER BY rank DESC;
```

SQL308	Add new board
Web resource	R302

```
INSERT INTO Board (description, name, id_creator, id_project)
VALUES ($description, $name, $id_creator, $id_project)
```

SQL309	Archive/close project
Web resource	*Resource not specified

```
UPDATE Project
SET project_state=$project_state
```

SQL310	Add new worker
Web resource	R209

```
INSERT INTO Project_team (id_user, id_project)
VALUES ($id_user, $id_project)
```

SQL311	Change project description
Web resource	*Resource not specified

```
UPDATE Project
SET description=$description
```

SQL312	Add project end date
Web resource	*Resource not specified

```
UPDATE Project
SET end_date=$end_date
```

SQL313	Change project picture
Web resource	*Resource not specified

```
INSERT INTO Project_picture (id_project, path)
VALUES ($id_project, $path)
```

SQL314	Send message to forum
Web resource	R211

```
INSERT INTO Message (message, id_user, id_project)
VALUES ($message, $id_user, $id_project)
```

SQL314	Trigger – no 2 boards on the same project can have the same name
Web resource	R302

```
CREATE FUNCTION check_exist_board() RETURNS TRIGGER AS
      $BODY$
      BEGIN
          IF EXISTS (SELECT name FROM Board WHERE
Board.id_project=$projectId AND NEW.name=name) THEN
              RAISE EXCEPTION 'This project already has a
board with that name'
          END IF;
          RETURN NEW;
      END
      $BODY$
      LANGUAGE plpgsql;
```

```
CREATE TRIGGER check_exist_board
    BEFORE INSERT ON Board
    FOR EACH ROW
        EXECUTE PROCEDURE check_exist_board();
```

SQL315	Trigger – a user can't be added to a project twice
Web resource	R209

```
CREATE FUNCTION check_exist_worker() RETURNS TRIGGER AS
    $BODY$
    BEGIN
        IF EXISTS (SELECT name FROM Project_team WHERE
Project_team.id_project=$projectId AND
NEW.id_worker=id_worker) THEN
            RAISE EXCEPTION 'This user already belongs to
this team'
        END IF;
        RETURN NEW;
    END
    $BODY$
    LANGUAGE plpgsql;

    CREATE TRIGGER check_exist_worker
        BEFORE INSERT ON Project_team
        FOR EACH ROW
            EXECUTE PROCEDURE check_exist_worker();
```

SQL316	Add user to contact list
Web resource	R111

```
INSERT INTO Contact (id_user, id_contact)
VALUES ($id_user, $id_contact)
```

SQL317	Trigger – can't add a user to contact list twice
Web resource	R111

```
CREATE FUNCTION check_exist_contact() RETURNS TRIGGER AS
    $BODY$
    BEGIN
        IF EXISTS (SELECT name FROM Contact WHERE
Contact.id_user=$userId AND NEW.id_contact=id_contact AND
NEW.id_contact=$userId) THEN
            RAISE EXCEPTION 'This user already is on your
contact list'
```

```
            END IF;
            RETURN NEW;
        END
        $BODY$
        LANGUAGE plpgsql;
        

        CREATE TRIGGER check_exist_contact
            BEFORE INSERT ON Contact
            FOR EACH ROW
                EXECUTE PROCEDURE check_exist_contact();
```

1.4. M043 Board pages

SQL301	Search board
Web resource	R215
SELECT id, name, description ts_rank_cd(textsearch, query) AS rank FROM Board, to_tsquery($search) AS query, to_tsvector(name || ' ' || description) AS textsearch WHERE query @@ textsearch\\ ORDER BY rank DESC;	

SQL401	Board tasks
Web resource	R213
SELECT * FROM Task WHERE Task.id_board=$boardId	

SQL402	Last 5 task updates
Web resource	* Resource not specified
SELECT TOP 5 Comment.id, File.id, Progress_update.id FROM Task INNER JOIN Comment ON Task.id=Comment.id_task INNER JOIN File ON Task.id=File.id_task INNER JOIN Task.id=Progress_update.id_task WHERE Task.id=$taskId	

SQL403	Search tasks
Web resource	R306
SELECT id, name, description ts_rank_cd(textsearch, query) AS rank FROM Task, to_tsquery($search) AS query, to_tsvector(name || ' ' || description) AS textsearch WHERE query @@ textsearch\\ ORDER BY rank DESC;	

SQL404	Add board meeting
Web resource	* Resource not specified
INSERT INTO Meeting (date, name, place, id_board) VALUES ($date, $name, $place, $id_board)	

SQL405	Add new worker to board
Web resource	R304
INSERT INTO Board_team (id_board, id_user) VALUES ($id_board, $id_user)	

SQL406	Send notification
Web resource	* Resource not specified
INSERT INTO Notification (id_user, notification) VALUES ($id_user, $notification)	

1.5. M054 Task pages

~~**SQL401**~~	~~Search task~~
~~**Web resource**~~	~~R307~~
~~SELECT id, name, description ts_rank_cd(textsearch, query) AS rank FROM Task, to_tsquery($search) AS query, to_tsvector(name || ' ' || description) AS textsearch WHERE query @@ textsearch\\ ORDER BY rank DESC;~~	

SQL501	Task information
Web resource	R405
SELECT * FROM Task WHERE Task.id=$taskId	

SQL502	Insert new progress update
Web resource	R404
INSERT INTO Progress_update (new_value, id_user, id_task) VALUES ($new_value, $id_user, $id_task)	

SQL503	Trigger – update task progress
Web resource	R404
CREATE FUNCTION update_task_progress() RETURNS TRIGGER AS $BODY$ BEGIN UPDATE Task SET Task.progress=Progress_update.new_value WHERE Task.id=Progress_update.id_task END	

```
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER update_task_progress
    AFTER INSERT ON Progress_update
    FOR EACH ROW
        Execute PROCEDURE update_task_progress();
```

SQL504	Upload new file
Web resource	R404
`INSERT INTO File (path, description, id_user, id_task)` `VALUES ($path, $description, $id_user, $id_task)`	

SQL505	Comment task
Web resource	R404
`INSERT INTO Comment (comment, id_user, id_task)` `VALUES ($comment, $id_user, $id_task)`	

SQL506	Archive/close task
Web resource	R404
`UPDATE Task` `SET task_state=$task_state`	

~~M05 Users~~

~~SQL501~~	~~Retrieve user's current projects~~
~~Web resource~~	~~R503~~
~~`SELECT username, full_name, e_mail FROM User`~~ ~~`  WHERE username LIKE %$search% OR full_name LIKE %$search%;`~~ ~~`ORDER BY username;`~~	

~~M06 User's projects~~

~~SQL601~~	~~Retrieve user's current projects~~
~~Web resource~~	~~R113~~
~~`SELECT *`~~ ~~`    FROM Project`~~ ~~`    INNER JOIN Project_team ON`~~ ~~`Project.id=Project_team.id_project`~~ ~~`    WHERE Project.id_coordinator=$user OR`~~ ~~`Project_team.id_user=$user`~~	

1.6. Homepage and administration

SQL601	All public projects
Web resource	* Resource not specified

```
SELECT name FROM Project
WHERE Project.privacy=1
```

SQL602	Search projects
Web resource	R701

```
SELECT id, name, description, start_date,
ts_rank_cd(textsearch, query) AS rank
FROM Project, to_tsquery($search) AS query, to_tsvector(name
|| ' ' || description) AS textsearch
WHERE query @@ textsearch\\ ORDER BY rank DESC;
```

SQL603	Set user as administrator
Web resource	R507

```
UPDATE User
SET administrator=$administrator
```

SQL604	Delete user
Web resource	R509

```
DELETE User
WHERE id=$id
```

SQL605	Delete project
Web resource	R607

```
DELETE Project
WHERE id=$id
```

2. Transactions

~~**T01**~~	~~Update task progress~~
~~**Isolation level**~~	
~~**Justification**~~	~~The task progress must be updated in two different tables, which means that if information is retrieved in the middle of the transaction, the information will not be coherent.~~

```
BEGIN TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMITTED

-- Insert new progress_update
INSERT INTO Progress_update (new_value, id_user, id_task)
VALUES ($new_value, $id_user, $id_task)

-- Update column progress in table Task
UPDATE Task
SET progress = $new_value
WHERE id = $id_task
```

```
COMMIT
```

T02	Send message to forum
Isolation level	
Justification	When a new message is sent (new instance in Message table), the last messages sent to the forum must be re-selected, or the user will not have access to updated information.

```
BEGIN TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMITTED

-- Insert new message
INSERT INTO Message (message, id_user, id_project)
VALUES ($message, $id_user, $id_project)

-- Select last messages
SELECT *
    FROM Message
    WHERE Message.id_project=$projectId AND Message.date >=
DATEADD(day, -7, GETDATE())

COMMIT
```

T03	Create task
Isolation level	
Justification	The board's tasks must be re-read after the addition of each new task, or the user will not have access to updated information.

```
BEGIN TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMITTED

-- Insert new task
INSERT INTO Task (description, name, id_creator, id_board, deadline,
budget)
VALUES ($description, $name, $id_creator, $id_project, $deadline,
$budget)

-- Select tasks from board
SELECT Task.id
    FROM Task
    WHERE Task.id_board=$boardId

COMMIT
```

T04	Create board
Isolation level	

Justification	The project's boards must be re-read after the addition of each new board, or the user will not have access to updated information.

```
BEGIN TRANSACTION
SET TRANSACTION ISOLATION LEVEL READ COMITTED

-- Insert new board
INSERT INTO Board (description, name, id_creator, id_project)
VALUES ($description, $name, $id_creator, $id_project)

-- Select boards from project and user
SELECT * FROM Board
    INNER JOIN Board_team ON Board.id=Board_team.id_board
    WHERE Board.id_project=$projectId AND Board_team.id_user=$id_user

COMMIT
```

T01	Change project state
Isolation level	SERIALIZABLE
Justification	When a project is archived (still exists, but can't be edited) or deleted, the state of all its boards and tasks must also change (to archived or deleted, respectively).

```
BEGIN TRANSACTION
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

-- Archive project
UPDATE Project
SET project_state=$states
WHERE id=$id

-- Archive project boards
UPDATE Board
SET board_state=$states
WHERE id_project=$id

-- Archive board tasks
UPDATE Task
SET task_state=$states
INNER JOIN Board ON Board.id=Task.id_board
WHERE Board.id_project=$id

COMMIT
```

T02	Delete user
Isolation level	SERIALIZABLE
Justification	When a user is deleted (removed from database), all the projects he coordinated are archived, as well as all their boards and projects.

```
BEGIN TRANSACTION
```

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

-- Delete user
DELETE FROM User
WHERE id=$userID

-- Archive project
UPDATE Project
SET project_state='Archived'
WHERE id_coordinator=$userID

-- Archive project boards
UPDATE Board
SET board_state='Archived'
INNER JOIN Project ON Board.id_project=Project.id
WHERE Porject.id_coordinator=$userID

-- Archive board tasks
UPDATE Task
SET task_state='Archived'
INNER JOIN Board ON Board.id=Task.id_board
    INNER JOIN Project ON Project.id=Board.id_project
WHERE Project.id_coordinator=$userID

COMMIT
```

T03	Get notifications
Isolation level	SERIALIZABLE READ ONLY
Justification	In the middle of the transaction, the insertion of new rows in the loan table can occur, which implies that the information retrieved in both selects is different, consequently resulting in a Phantom Read. It's READ ONLY because it only uses Selects.

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

-- Get number of unread notifications
SELECT COUNT id
    FROM Notification
    WHERE Notification.id_user=$user AND Notification.read=0

-- Get notifications
SELECT * FROM Notification
    WHERE Notification.id_user=$user

COMMIT
```

Changes made to last submission:

- Added more queries to every module, in order to increase complexity and cover all necessary queries
- Dropped previous transactions, that were unnecessary
- Added 3 new transactions

GROUP1734, 21/04/2018:

- Maria Inês Gonçalves, up201402784@fe.up.pt
- Maria Teresa Valério, up201405655@fe.up.pt
- Sara Gomes, up201405085@fe.up.pt