GITHUB Repository

TEAM 2

Team Members:

Devanshi Trivedi (013530530)

Keerthi Akella (013858819)

Nithin Gollanapally (013820599)

Thinh Manh(012050571)

Rajesh Thummala (013829179)

Siddarth Varanasi (013854295)

Choice of Database project:

This application is a version control tool for coder and developer. Here, people can come together and work on a same project individually. They will be able to make edits for the code that has been shared. When they finished their edit, they can merge all the version into one.

Many times, it happens that the programmer wants to revert to a change he made. GitHub makes this possible by providing a feature called commits. Here a commit is done whenever some changes are made which can be accessed later.
Coders can also follow each other and can be inspired by others work. They can post their projects which can be accessed by everyone.

In this application, MySQL is used as Database engine. JDBC, JAVA, Log 4J are database application technologies. However, there will be no frameworks will be used in the project. Besides, Java is the only programming language, and JDBC 6.0 is the database access technology.

Architecture Block diagram:

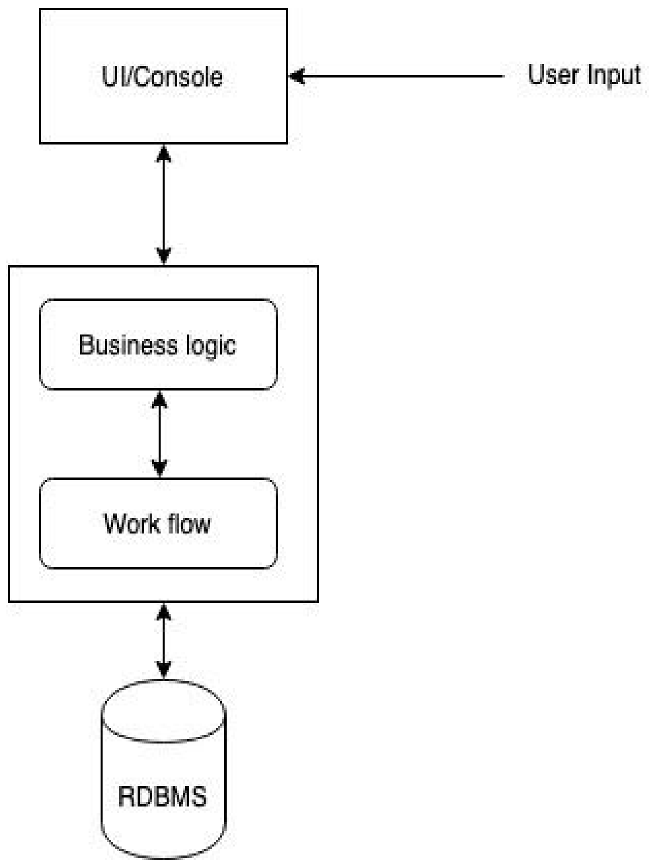


Figure1: Architecture Block Diagram of the project

Functionalities

1) ### SignUp /Save the hashed password

   A user can sign up for a Github account with a password that will be encrypted while submitting the form.

2) ### Login /Decrypt the password

   A user logs in to his account with his password to access his data(repository). In this process, user credentials are encrypted and validated with the database hashed password.

3) ### Create a repository / View all repository

   A user can create a repository where he can create number of branches and can store his project files.

4) ### Create a branch / View all branches

   The repository members can create as many branches of the repository as he wants. Usually, each repository member is assigned a branch. The code in each branch is separate and so all members can work on same project without editing each other's work.

5) ### Create a project file / View all projects for a branch

   Each repository has project files, which may contain the project related files and source code.

6) ### Select a branch

   a) #### Commit in a branch /View all commits for that branch

      Users can commit their works into the branch every time they add a new file. If the user wants to revert to a change he made and also can add code, GitHub makes this possible under this feature.

   b) #### Pull_request from a branch /View all pull_requests for that branch

      Users can request to pull the data to their system by this feature.

c) Comments in that branch /View all comments for that branch

Users who commit to the branch with a comment of changes they made.

7) Follow Someone /View all the followers

Each user can follow one or more users, and they can view their project files. Each user can see the list of people following them.

8) Logout

User after logging out of the account, gets reverted back to the login page.

ER Diagram

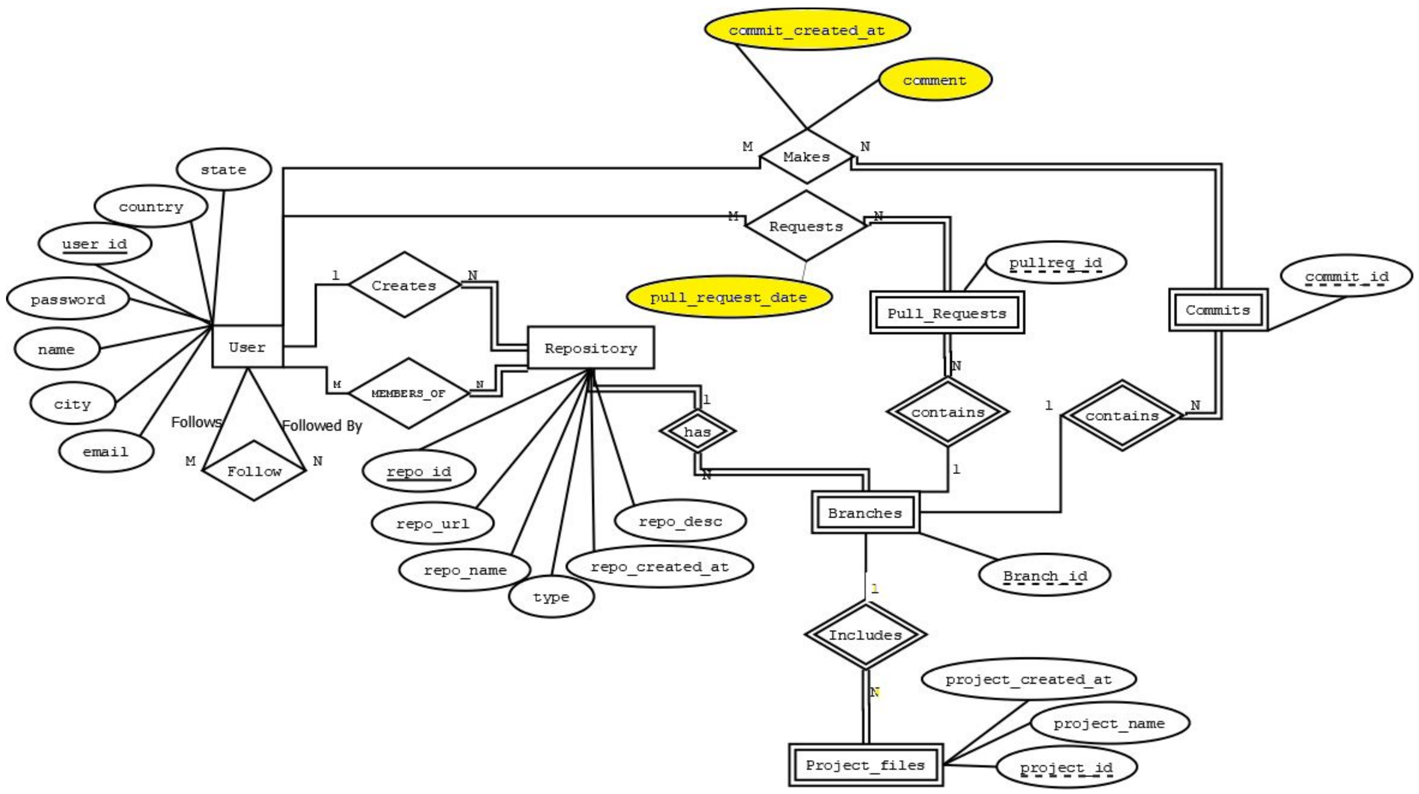


Figure2: ER Diagram with changes highlighted

Specification of each DB object and its meaning:

User: This entity to store user information such as name, city, email, password, country, state, user_id. Each user has their unique id when they create an account.

Repository: This entity has create and member_of relationship with user entity. The create relationship allow user to create repositories and the user can be a member of multiple repositories.

Branches: It is an entity that depends on Repository entity, This entity allow users to create as many branches in the repository as they wish and related branches can be belong to only belong to a specific Repository entity. Each branch has their unique branch id. These branch_id can be duplicate for different repository.

Project_files: It is an entity that depends on branches entity and Repository entity. This allow users share their their project files. The project_files entity has project_created_at, project_name, and unique project_id attributes. Project_id can be duplicated among other project_id in different branches.

Pull_Request: It is an entity that is not only depends on users entity but also branches. The user send a request for pull_request. Each pull request entity can have a unique pullreq_id. Pullreq_id can be duplicate in different branches.

Commit: It is an entity that allow user to commit their changes into branches. Each commit has unique commit_id, and commit_id can be duplicated in different branches.

Functionality dependencies:

Users:

<u>User_ID</u>	password	name	email	city	state	country

User ID → {password, name, email, city, state, country}

Repository:

<u>User_ID</u>	<u>Repo_ID</u>	repo _URL	repo_name	repo_desc	repo_created_at

User ID, Repo ID → {repo_eurl, repo_name, type, repo_desc, repo_created_at}

Branches:

<u>Repo_ID</u>	<u>Branch_ID</u>	Branch_name

Repo ID, Branch ID → { branch_name}

Project_files:

<u>Repo_ID</u>	<u>Branch_ID</u>	<u>Project_ID</u>	Project_name	Project_created_at

Repo_ID, branch_ID, project_ID → { project name, project created at}

Makes:

<u>User_ID</u>	<u>Repo_ID</u>	<u>Branch_ID</u>	<u>Commit_ID</u>	Comment	Commit_creted_at

User_ID, Repo_ID, Branch_ID, Commit_ID → {comment, commit_created_at}

Requests:

<u>User_ID</u>	<u>Repo_ID</u>	<u>Branch_ID</u>	<u>Pull_req_ID</u>	pull_req_date

User_ID, Repo_ID, Branch_ID, Pull_req_ID → { pull_req_date}

Schema Diagram:

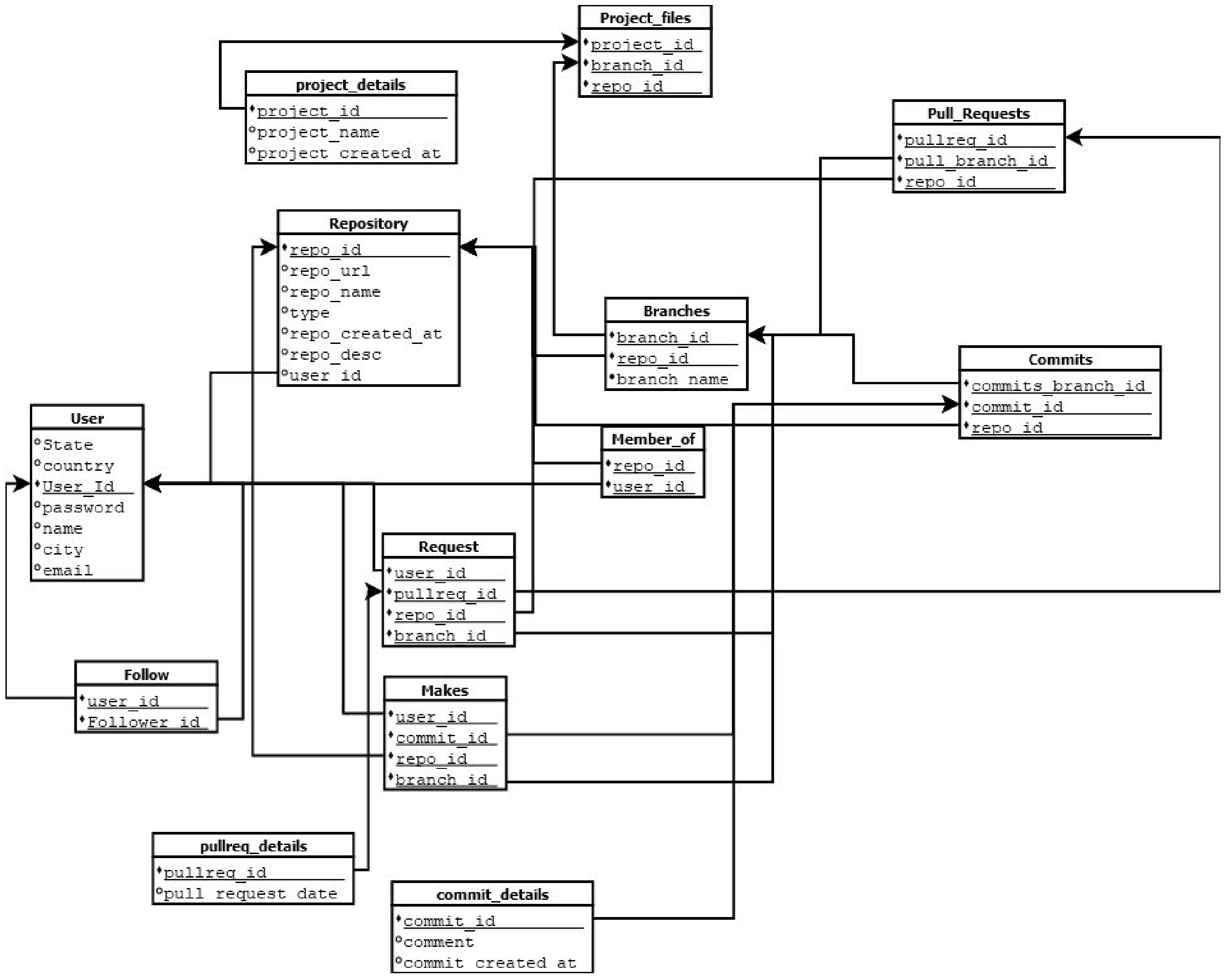


Figure3: Schema Diagram of Database

Normalization

1. user: user_id(pk), password, name, email, city, state, country

   The table has one primary key which is user_id, User ID → {password, name, email, city, state, country}, and there is no multivalued attributes or nested relation. Thus, this table is 1st normal form(NF). The table is also in 2nd normal form because it is 1NF and there is no partial dependency in the table. The table is 3rd normal form because it is 2NF and the attributes has no transitive dependencies.

2. followers(user_id , follower)

   This table contains only composite primary key so it is the 1st NF. Since it is 1st NF, and there is no partial dependencies so it is 2nd NF. Lastly, the table is also 3rd NF because it is 2nd NF and it has no transitive dependencies

3. Repository (user_id, repo_id(pk), repo_url,repo_name, repo_desc, repo_created_at)

   This table has user_id as the foreign key and repo_id as primary key, repo_id->{ user_id, repo_url,repo_name, type, repo_desc, repo_created_at}, and there is no multivalued attributes or nested relation. Therefore, this is the 1st NF. Since it is 1st NF, and there is no partial dependencies so it is 2nd NF. The table is also 3rd NF because table is 2nd NF and it is fully dependencies with no transitive dependencies

4. Members_of:(user_id(pk), repo_id(pk))

   This table has two composite primary key which is the same as followers table. Thus, it is 3rd NF.

5. branches (repo_id, branch_id, branch_name)

   This table also has two composite primary key which are repo_id (foreign key) and branch_id, and there is no multivalued or nested relation. Hence, it is the 1st NF. The table is also 2nd NF because it is 1st NF and branch name depends on both repo_id and branch_id so it is 2nd NF. The table is 2nd NF, branch_name attribute fully depends on both composite primary key, and there is no transitive dependencies. Thus, it is the 3rd NF.

6. Project_files (repo_id, branch_id, project_id, project_name, project_created_at)

BREAKS DOWN TO
(
    project_files: repo_id, branch_id, project_id

project_details: project_id, project_name, project_created_at
)

The project_files has three composite primary key which are repo_id(foreign key), branch_id, and project_id. The project_file table has no multivalued attributes or nested relation. Thus, project_file table is 1st NF. This table is 1st NF , and project_name and project_created_at are depend on all three primary keys. Thus, it is 2NF. However, these attributes have transitive dependencies. Therefore, it is necessary to break down the table into 2 sub tables which are project_files and project_details. In the project_file table, the schema remains same with three composite primary key (repo_id, branch_id, and project_id) since this table contains only three primary so it is 3rd NF. In the project_details schema, it has only project_id as primary key, and there are no multivalued or composite attributes. Thus this table is 1st NF, and the project_name and project_created_at depends only on prject_id. Hence, it is 2nd NF. Because this table is 2nd NF, and these attributes is fully depends on primary key and there is no transitive dependencies. As a result, project_details schema is also in 3rd NF

7. makes: user_id, repo_id, branch_id, commit_id, comment, commit_created_at

BREAKS DOWN TO
(
makes: user_id, repo_id, branch_id, commit_id
commit_details: commit_id, comment, commit_created_at
)

Makes schema has the same problem as project_file schema since the schema is 1st NF because it has four composite primary keys and there is no multivalued attributes and nested relation. Makes schema qualify for 1st NF relation, and the attributes depends on the composite primary key. Thus, it is in 2 NF. However, the makes schema is not fully dependencies. So,Makes is broken down into two schema which are makes and commit_details. The makes schema keeps all the primary from original schema, and the table has no extra attributes. Thus, it is in 3rd NF. For the commit_details schema, it has one commit_id primary key, and the attributes are not multivalued attributes and no nested relation. Hence, it is 1st NF, and these attributes depends only on commit_id primary key. Thus, it is 2nd NF. The schema is 2nd NF, and there is no transitive dependencies in this table which means comment and commit_created_at are both directly depend on commit_id. As a result, it is in 3rd NF.

8. requests: user_id, repo_id, branch_id, pullreq_id, pull_request_date

BREAKS DOWN TO
(

requests: user_id, repo_id, branch_id, pullreq_id
pullreq_details: pullreq_id, pull_request_date
)

Requests schema has four composite primary keys and there is no multivalued attribute or nested relation. Thus ,it is 1st NF. The requests schema is 2nd NF because it first satisfied 1NF and pull_request_date depends only on three primary key. Nevertheless, the requests schema is not fully dependencies and they are transitive dependencies. Eventually, the schema is broken down into two part requests and pullreq_details. Since requests schema only contains four composite primary key. Therefore it is in 3rd NF. The pullreq_details schema has pullreq_id primary key, and pull_request_date is not a multivalued attribute or relation schema. Therefore, pullreq_details schema is 1NF. Pullreq_details is 1NF, and pull_request_date depends only on pullreq_id. Therefore, it is 2NF. The pullreq_details is 2NF and there is no transitive dependencies. Thus, it is also 3NF schema.

9. commits: repo_id, branch_id, commit_id

Commits contains 3 composite primary keys. There is no multivalued attributes or nested relation. Thus, it is 1NF. Besides, there are only primary key so there is no partial dependencies in the table, Thus it is 2NF. In addition, there are only three primary key so it is full dependencies. So, it is in 3NF.

10. pull_requests: repo_id, branch_id, pullreq_id

Pull_requests schema has 3 composite primary keys. There is no multivalued attributes or nested relation. Thus, it is 1NF. Moreover, there is no partial dependencies in the table. Thus it is 2NF. In addition, shema is fully dependencies. Hence, it is 3NF.

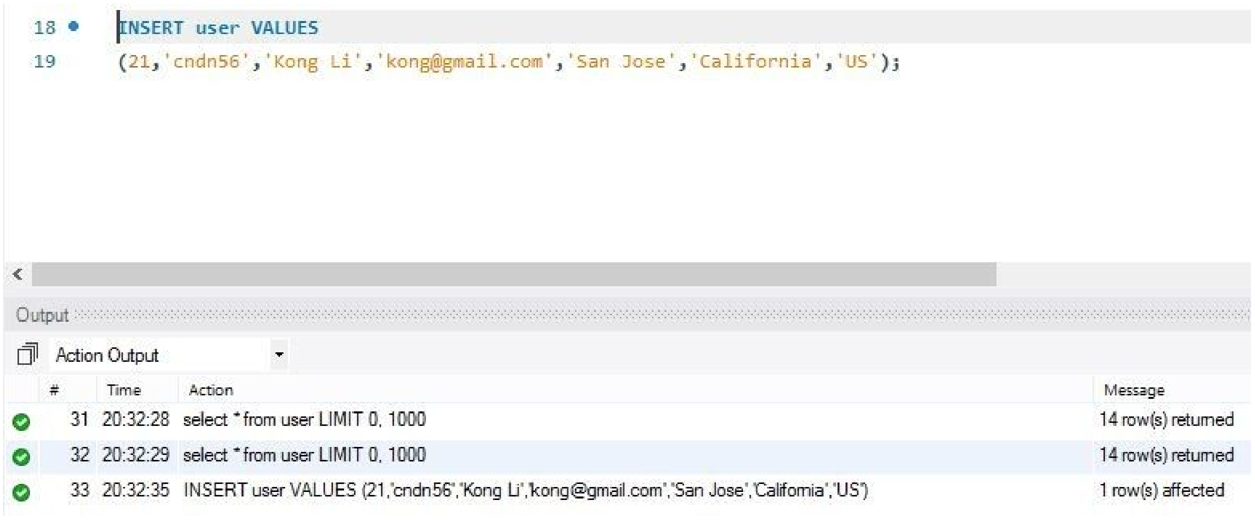
Final major areas/components/tasks:

- Project ER Diagram: Equal contribution by all the team members.
- Implementation, coding and testing of the following major functionalities:

Milestone	Planned End Date	Actual End Date	Resource
Login page/Logout page	03/30/2019	04/01/2019	Devanshi
View Repositories page	04/05/2019	04/06/2019	Siddarth
Create Repository page	04/10/2019	04/11/2019	Nithin
View Followers page	04/15/2019	04/16/2019	Rajesh
View Following page	04/20/2019	04/21/2019	Keerthi
Follow a user page	04/25/2019	04/26/2019	Thinh

Screenshots of Major operations:

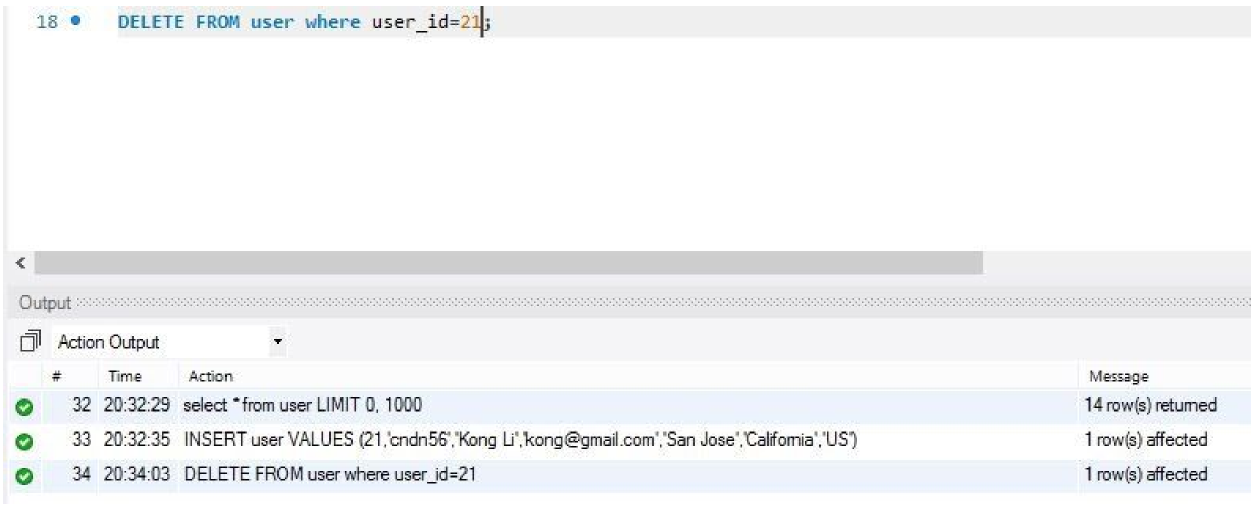
Insert into User query



```
INSERT user VALUES
(21,'cndn56','Kong Li','kong@gmail.com','San Jose','California','US');
```

```
Output
Action Output
 #   Time      Action                                                                           Message
 31  20:32:28  select * from user LIMIT 0, 1000                                                 14 row(s) returned
 32  20:32:29  select * from user LIMIT 0, 1000                                                 14 row(s) returned
 33  20:32:35  INSERT user VALUES (21,'cndn56','Kong Li','kong@gmail.com','San Jose','California','US')   1 row(s) affected
```

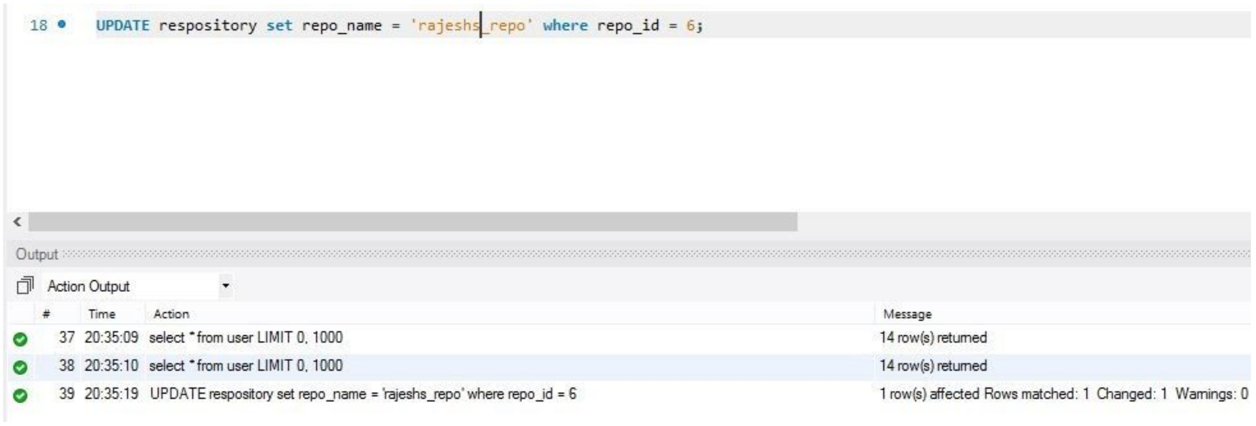
Delete from user query



```
DELETE FROM user where user_id=21;
```

```
Output
Action Output
 #   Time      Action                                                                           Message
 32  20:32:29  select * from user LIMIT 0, 1000                                                 14 row(s) returned
 33  20:32:35  INSERT user VALUES (21,'cndn56','Kong Li','kong@gmail.com','San Jose','California','US')   1 row(s) affected
 34  20:34:03  DELETE FROM user where user_id=21                                                1 row(s) affected
```

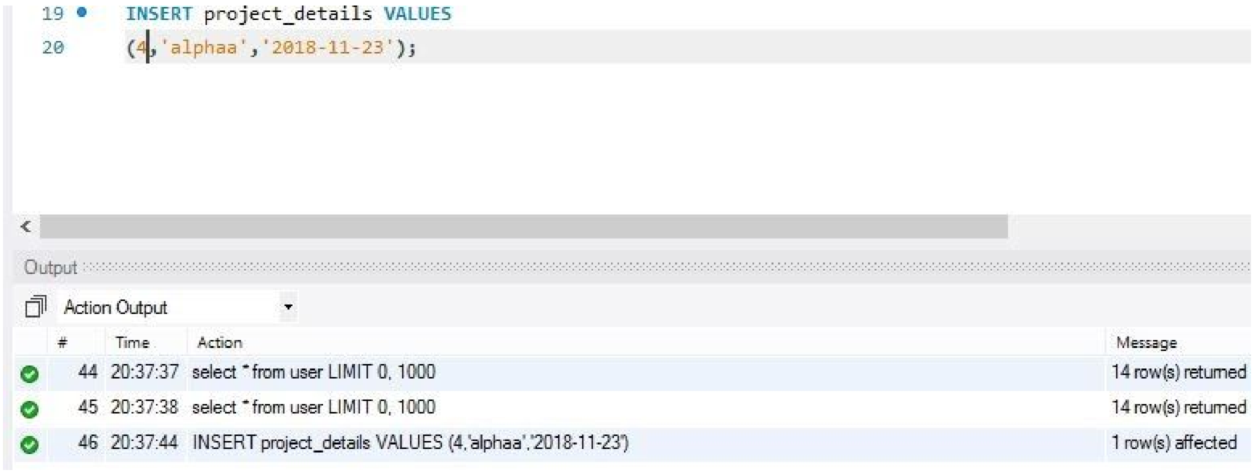
Update repository name query



```sql
UPDATE respository set repo_name = 'rajeshs_repo' where repo_id = 6;
```

```
#   Time      Action                                                          Message
37  20:35:09  select * from user LIMIT 0, 1000                                14 row(s) returned
38  20:35:10  select * from user LIMIT 0, 1000                                14 row(s) returned
39  20:35:19  UPDATE respository set repo_name = 'rajeshs_repo' where repo_id = 6   1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
```

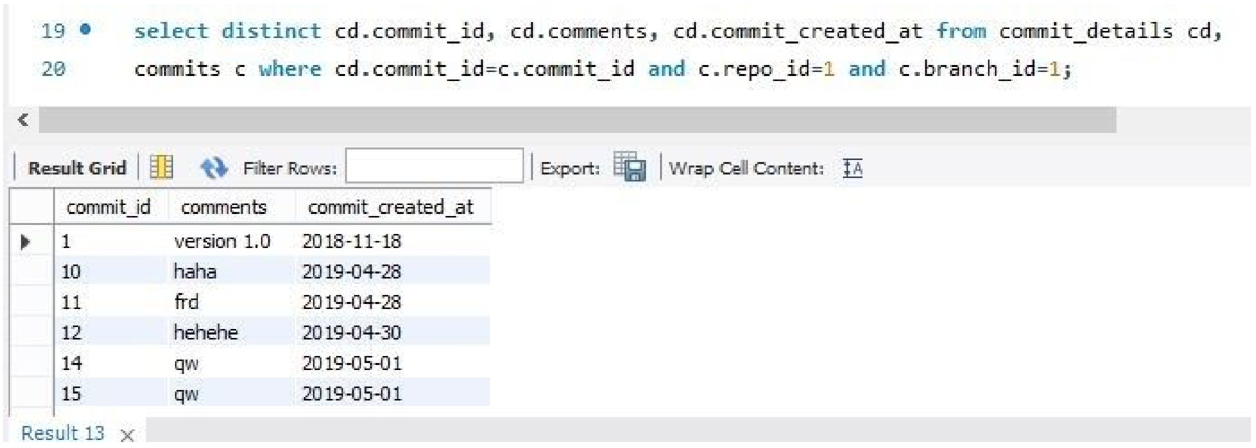
Insert project details query



```sql
INSERT project_details VALUES
(4,'alphaa','2018-11-23');
```

```
#   Time      Action                                                 Message
44  20:37:37  select * from user LIMIT 0, 1000                       14 row(s) returned
45  20:37:38  select * from user LIMIT 0, 1000                       14 row(s) returned
46  20:37:44  INSERT project_details VALUES (4,'alphaa','2018-11-23')  1 row(s) affected
```

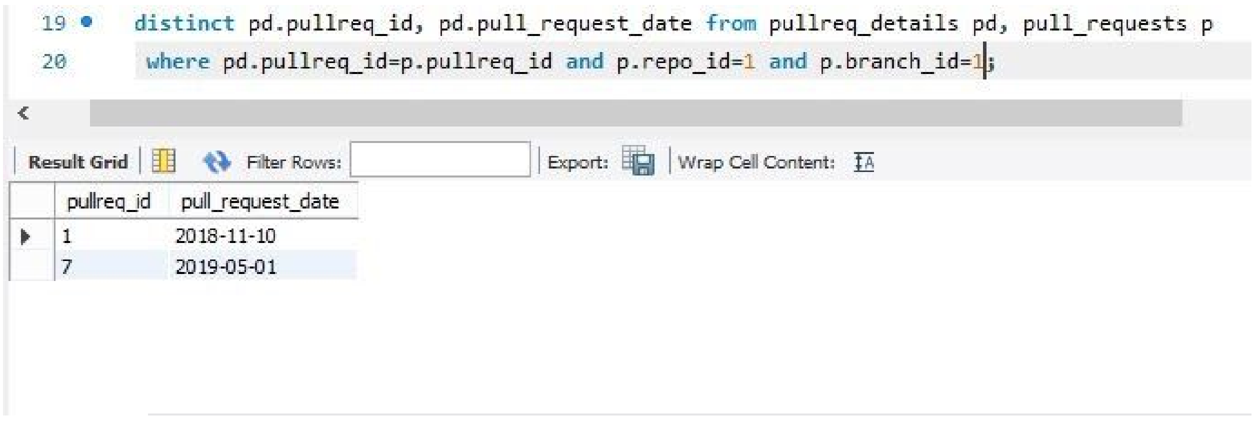
View Commits query



```sql
select distinct cd.commit_id, cd.comments, cd.commit_created_at from commit_details cd,
commits c where cd.commit_id=c.commit_id and c.repo_id=1 and c.branch_id=1;
```

commit_id	comments	commit_created_at
1	version 1.0	2018-11-18
10	haha	2019-04-28
11	frd	2019-04-28
12	hehehe	2019-04-30
14	qw	2019-05-01
15	qw	2019-05-01

View Pull requests query



```
distinct pd.pullreq_id, pd.pull_request_date from pullreq_details pd, pull_requests p
where pd.pullreq_id=p.pullreq_id and p.repo_id=1 and p.branch_id=1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

```
pullreq_id  pull_request_date
1           2018-11-10
7           2019-05-01
```

• Test cases:

Scenarios: Navigation					
Test case ID	**Test cases**	**Expected result**	**Actual result**	**Positive/Negative**	**Pass/ Fail**
1	Test case to verify if a screen with signup, login and Admin login is displayed	A screen with signup, login and Admin login should be displayed	A screen with signup, login and Admin login is displayed	Positive	Pass
2	Test case to verify if a user can navigate to the home screen on selecting the 'Go to home' selection.	A user should be able to navigate to the home screen on selecting the 'Go to home' selection.	A user is able to navigate to the home screen on selecting the 'Go to home' selection.	Positive	Pass

3	Test case to verify if a user can navigate to the 'Signup' screen from the home screen containing subsequent screens asking for user input of their name, email ID, password, city, state and country.	A user should be able to navigate to the 'Signup' screen from the home screen containing subsequent screens asking for user input of their name, email ID, password, city, state and country.	A user is able to navigate to the 'Signup' screen from the home screen containing subsequent screens asking for user input of their name, email ID, password, city, state and country.	Positive	Pass
4	Test case to verify if a user can navigate to the 'Login' screen from the home screen containing subsequent screens asking for user input of their registered email ID and password.	A user should be able to navigate to the 'Login' screen from the home screen containing subsequent screens asking for user input of their registered email ID and password.	A user is able to navigate to the 'Login' screen from the home screen containing subsequent screens asking for user input of their registered email ID and password.	Positive	Pass
5	Test case to verify if an admin can navigate to the 'Admin Login' screen from the home screen containing subsequent screens asking for user input of their registered email ID and password.	An admin should be able to navigate to the 'Admin Login' screen from the home screen containing subsequent screens asking for user input of their	An admin is able to navigate to the 'Admin Login' screen from the home screen containing subsequent screens asking for user input of	Positive	Pass

		registered email ID and password.	their registered email ID and password.		
6	Test case to verify if an admin can navigate to subsequent screens containing 'Delete User', 'Delete Repository', 'Rename Repo Name' and 'Rename Branch Name'.	An admin should be able to navigate to subsequent screens containing 'Delete User', 'Delete Repository', 'Rename Repo Name' and 'Rename Branch Name'.	An admin is able to navigate to subsequent screens containing 'Delete User', 'Delete Repository', 'Rename Repo Name' and 'Rename Branch Name'.	Positive	Pass
7	Test case to verify if a user can navigate to a screen containing View Branches, Create a Branch, Go to Home or Logout from the 'View Repositories' screen.	A user should be able to navigate to a screen containing View Branches, Create a Branch, Go to Home or Logout fro the 'View Repositories' screen.	A user is able to navigate to a screen containing View Branches, Create a Branch, Go to Home or Logout fro the 'View Repositories' screen.	Positive	Pass
8	Test case to verify if a user can navigate to a screen containing View Project Files, Create a Project File, View commits, Go to Home or Logout from the 'View Branches' screen.	A user should be able to navigate to a screen containing View Project Files, Create a Project File, View commits, Go to Home or	A user is able to navigate to a screen containing View Project Files, Create a Project File, View commits, Go to Home or	Positive	Pass

		Logout from the 'View Branches' screen.	Logout from the 'View Branches' screen.		
9	Test case to verify if a user is able to navigate to a screen containing Commit, Pull request, Go to Home or Logout from the 'View Project Files' screen.	A user should be able to navigate to a screen containing Commit, Pull request, Go to Home or Logout from the 'View Project Files' screen.	A user is able to navigate to a screen containing Commit, Pull request, Go to Home or Logout from the 'View Project Files' screen.	Positive	Pass
10	Test case to verify if the user is able to navigate to a screen containing 'enter your comment' from the 'Commit' screen.	The user should be able to navigate to a screen containing 'enter your comment' from the 'Commit' screen.	The user is able to navigate to a screen containing 'enter your comment' from the 'Commit' screen.	Positive	Pass
11	Test case to verify if a user is able to navigate to a screen containing 'Enter your Project File name' from the 'Create a Project file' screen.	A user should be able to navigate to a screen containing 'Enter your Project File name' from the 'Create a Project file' screen.	A user is able to navigate to a screen containing 'Enter your Project File name' from the 'Create a Project file' screen.	Positive	Pass
12	Test case to verify if a user is able to navigate to a screen containing view commits on selecting 'View Commits'	A user should be able to navigate to a screen containing	A user is able to navigate to a screen containing view commits	Positive	Pass

		view commits on selecting 'View Commits'	on selecting 'View Commits'		
13	Test case to verify if a user is able to navigate to a screen containing 'Enter your branch name' screen from the 'Create a branch' screen.	A user should be able to navigate to a screen containing 'Enter your branch name' screen from the 'Create a branch' screen.	A user is able to navigate to a screen containing 'Enter your branch name' screen from the 'Create a branch' screen.	Positive	Pass
14	Test case to verify if a user is able to navigate to subsequent screens containing 'Enter your Rep name' and 'Enter your Repo Desc' from the 'Create Repository' screen.	A user should be able to navigate to subsequent screens containing 'Enter your Rep name' and 'Enter your Repo Desc' from the 'Create Repository' screen.	A user is able to navigate to subsequent screens containing 'Enter your Rep name' and 'Enter your Repo Desc' from the 'Create Repository' screen.	Positive	Pass
15	Test case to verify if a user is able to navigate to a screen containing their followers and whom they are following from the 'View followers' and 'View following' screens respectively.	A user should be able to navigate to a screen containing their followers and whom they are following from the 'View followers' and 'View following' screens	A user is able to navigate to a screen containing their followers and whom they are following from the 'View followers' and 'View following' screens	Positive	Pass

		respectively.	respectively.		
16	Test case to verify if a user is able to navigate to a screen containing 'Enter the email of the user you want to follow:' from the 'Follow a user' screen.	A user should be able to navigate to a screen containing 'Enter the email of the user you want to follow:' from the 'Follow a user' screen.	A user is able to navigate to a screen containing 'Enter the email of the user you want to follow:' from the 'Follow a user' screen.	Positive	Pass
17	Test case to verify if a user is able to navigate to a screen containing the logout message from the 'Logout' screen.	A user should be able to navigate to a screen containing the logout message from the 'Logout' screen.	A user is able to navigate to a screen containing the logout message from the 'Logout' screen.	Positive	Pass
Scenario: Functionality					
Test case ID	**Test cases**	**Expected result**	**Actual result**	**Positive/Negative**	**Pass/ Fail**
18	Test case to verify if a user can signup using their name, email ID, password, city, state and a country.	A user should be able to signup using their name, email ID, password, city, state and a country.	A user is able to signup using their name, email ID, password, city, state and a country.	Positive	Pass
19	Test case to verify if a user can login using their registered email ID and password.	A user should be able to login using their registered email ID and password.	A user is able to login using their registered email ID and password.	Positive	Pass

20	Test case to verify if an admin can login using their registered email ID and password.	An admin should be able to login using their registered email ID and password.	An admin is able to login using their registered email ID and password.	Positive	Pass
21	Test case to verify if an admin can delete a user after logging into the admin login.	An admin should be able to delete a user after logging into the admin login.	An admin is able to delete a user after logging into the admin login.	Positive	Pass
22	Test case to verify if an admin can delete a repository after logging into the admin login.	An admin should be able to delete a repository after logging into the admin login.	An admin is able to delete a repository after logging into the admin login.	Positive	Pass
23	Test case to verify if an admin can rename a repository name after logging into the admin login.	An admin should be able to rename a repository name after logging into the admin login.	An admin is able to rename a repository name after logging into the admin login.	Positive	Pass
24	Test case to verify if an admin can rename a branch name after logging into the admin login.	An admin should be able to rename a branch name after logging into the admin login.	An admin is able to rename a branch name after logging into the admin login.	Positive	Pass
25	Test case to verify if a user can commit Project files by entering their comment on selecting 'Commit'.	A user should be able to commit Project files by entering their comment on	A user is able to commit Project files by entering their comment on selecting	Positive	Pass

		selecting 'Commit'.	'Commit'.		
26	Test case to verify if the user can perform a pull request on the project files on selecting 'Pull Request'.	The user should be able to perform a pull request on the project files on selecting 'Pull Request'.	The user is able to perform a pull request on the project files on selecting 'Pull Request'.	Positive	Pass
27	Test case to verify if a user can View Project files in a branch on selecting the 'View Project Files' selection.	A user should be able to View Project files in a branch on selecting the 'View Project Files' selection.	A user is able to View Project files in a branch on selecting the 'View Project Files' selection.	Positive	Pass
28	Test case to verify if a user can create Project files in a branch by entering the project file name on selecting the 'Create Project Files' selection.	A user should be able to create Project files in a branch by entering the project file name on selecting the 'Create Project Files' selection.	A user is able to create Project files in a branch by entering the project file name on selecting the 'Create Project Files' selection.	Positive	Pass
29	Test case to verify if a user can view existing Commits in a branch on selecting 'View Commits'.	A user should be able to view existing Commits in a branch on selecting 'View Commits'.	A user is able to view existing Commits in a branch on selecting 'View Commits'.	Positive	Pass

30	Test case to verify if a user can view existing branches in a repository on selecting 'View branches'.	A user should be able to view existing branches in a repository on selecting 'View branches'.	A user is able to view existing branches in a repository on selecting 'View branches'.	Positive	Pass
31	Test case to verify if a user can create a branch by entering the branch name in a repository on selecting 'Create a branch'.	A user should be able to create a branch by entering the branch name in a repository on selecting 'Create a branch'.	A user is able to create a branch by entering the branch name in a repository on selecting 'Create a branch'.	Positive	Pass
32	Test case to verify if a user can view existing repositories on selecting 'View Repositories'.	A user should be able to view existing repositories on selecting 'View Repositories'.	A user is able to view existing repositories on selecting 'View Repositories'.	Positive	Pass
33	Test case to verify if a user can create a repository by entering the repository name and repository description on selecting 'Create Repository'.	A user should be able to create a repository by entering the repository name and repository description on selecting 'Create Repository'.	A user is able to create a repository by entering the repository name and repository description on selecting 'Create Repository'.	Positive	Pass
34	Test case to verify if a user can view their followers on selecting 'View followers'.	A user should be able to view their followers on selecting	A user is able to view their followers on selecting	Positive	Pass

		'View followers'.	'View followers'.		
35	Test case to verify if a user can view whom they are following on selecting 'View following'.	A user should be able to view whom they are following on selecting 'View following'.	A user is able to view whom they are following on selecting 'View following'.	Positive	Pass
36	Test case to verify if a user can follow another user by entering the email of the user that they want to follow on selecting 'Follow a user'.	A user should be able to follow another user by entering the email of the user that they want to follow on selecting 'Follow a user'.	A user is able to follow another user by entering the email of the user that they want to follow on selecting 'FollowA user'.	Positive	Pass
37	Test case to verify if a user can logout of their account on selecting 'Logout'.	A user should be able to logout of their account on selecting 'Logout'.	A user is able to logout of their account on selecting 'Logout'.	Positive	Pass
Scenario: Negative					
Test case ID	**Test cases**	**Expected result**	**Actual result**	**Positive/Negative**	**Pass/ Fail**
38	Test case to verify if a user cannot login if they did not use their registered email ID and password.	A user shouldn't be able to login when they do not use their registered email ID and password.	A user is unable to login when using they don’t use their registered email ID and password.	Negative	Pass

39	Test case to verify if a user cannot login with their credentials into an admin account.	A user should not be able to login with their credentials into an admin account.	A user is unable to login with their credentials into an admin account.	Negative	Pass
40	Test case to verify if an admin cannot login with wrong credentials into an admin account.	An admin should not be able to login with wrong credentials into an admin account.	An admin is not able to login with wrong credentials into an admin account.	Negative	Pass

Test plan execution:

1. Overview

1.1. Purpose

The purpose of this document is to define:

- The test scope, focus areas and objectives
- The test schedule and major milestones
- The test deliverables

1.2. Scope

This document details the testing that will be performed by the project team2 for the GitHub Repository project. It defines the overall testing requirements and provides an integrated view of the project test activities. Its purpose is to document:

- What will be tested;
- How testing will be performed

2. Testing Summary

2.1. Scope of Testing

2.1.1. In scope

- *Navigation testing of the application*
- *Functionalities testing*
- *Negative testing*

3. Test Strategy

3.1. Test Type & Approach

Test Type	Objectives
Navigation Testing	The objectives are to verify that the application: · Meets the defined requirements; · Interfaces function correctly;
Functional testing	· Performs and functions accurately; · Correctly handles error conditions;

Negative testing	· Ensures exception handling and errors.

3.2. Test Execution Schedule

Milestone	Planned End Date	Actual End Date	Resource
Login page/Logout page	**03/30/2019**	**04/01/2019**	**Devanshi**
View Repositories page	**04/05/2019**	**04/06/2019**	**Siddarth**
Create Repository page	**04/10/2019**	**04/11/2019**	**Nithin**
View Followers page	**04/15/2019**	**04/16/2019**	**Rajesh**
View Following page	**04/20/2019**	**04/21/2019**	**Keerthy**
Follow a user page	**04/25/2019**	**04/26/2019**	**Thinh**

3.3. Testing Tools

Detail the tools to be used for testing.
For example:
The following tools will be used for testing:

Process	Tool
Test case creation	Microsoft Word
Test case tracking	Microsoft Excel
Test case execution	Manual
Test case management	Microsoft Excel
Defect management	Microsoft Excel

4. References

The following documents have been used to assist in creation of this document.

#	Document name	Version	Comments
1	https://webcache.googleusercontent.com/search?q=cache:GB2JYc0Ohe8J:https://strongqa.com/uploads/document/doc/29/test-plan-template-04.docx+&cd=1&hl=en&ct=clnk&gl=us	1.0	Template for test plan execution.

5. Points of Contact

The following people can be contacted in reference to this document

Primary Contact	
Name	Nithin
Title/Organisation	SJSU

Phone	669-252-9454
Email	Nithin.gollanapally@sjsu.edu
Secondary Contact	
Name	Siddarth
Title/Organisation	SJSU
Phone	669-252-9549
Email	Siddarth.varanasi@sjsu.edu

Project postmortem

- o issues uncovered//
- o implement something differently// password encryption
- o potential improvements// UI
- o etc.