

DATABASE MANAGEMENT SYSTEM Project

My-Git

PES1UG19CS579

VISHWAS R

PES1UG19CS548

UTHPAL P

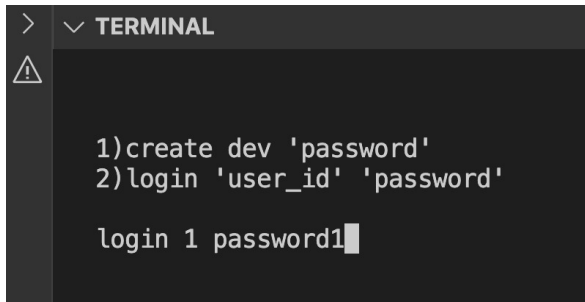
PES1UG19CS534

T R SUDHARSHAN

Front End Implementation:

User provided with a Command Line Interface as the front-end.

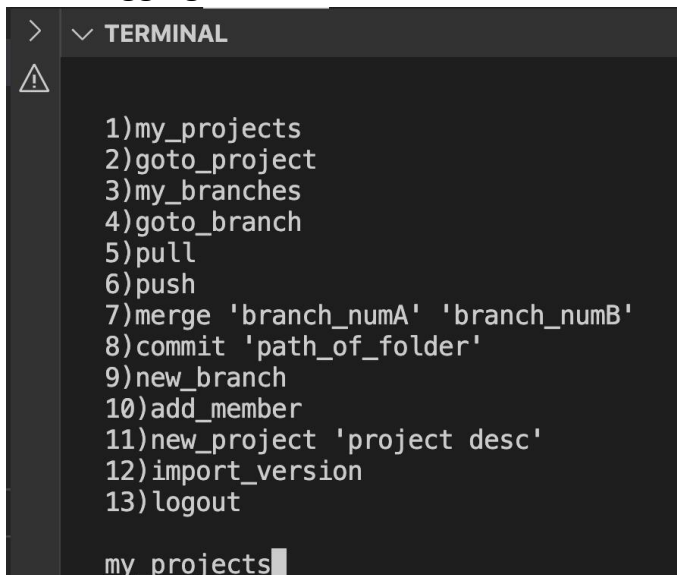
- Initial/ First screen:
User can either login with the registered credentials or create a new developer.



```
> ∨ TERMINAL
!
1)create dev 'password'
2)login 'user_id' 'password'

login 1 password1
```

- After logging in with the correct credentials, the options provided are:



```
> ∨ TERMINAL
!
1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout

my_projects
```

- Entering my_projects will display all the different projects the developer is involved in:

```

your_local_repo_id-> 4 remote_repo_id-> 1 Project Description -> proj_desc_1

1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout

goto_project 1

```

- Navigate into any of the projects the developer is given access to by entering the remote repo id of that project listed:

Now you're in project 1 with local repo id 4

```

1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout

my_branches

```

- After navigating into a project, the developer can add members to that project or can further navigate into different branches present in their local repository:

```

Branches present are
1 2
1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout

goto_branch 1

```

- After navigating into a branch, the developer can make new commits by providing the folder destination containing the files to be committed into that branch:

```
Current branch 1

1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout

commit '/Users/vishwas/Downloads/Front_end_implementation/new'
```

- The developers can even create new branches from the current branch:

```
Branch Successfull and new branch num = 5

1)my_projects
2)goto_project
3)my_branches
4)goto_branch
5)pull
6)push
7)merge 'branch_numA' 'branch_numB'
8)commit 'path_of_folder'
9)new_branch
10)add_member
11)new_project 'project desc'
12)import_version
13)logout
```

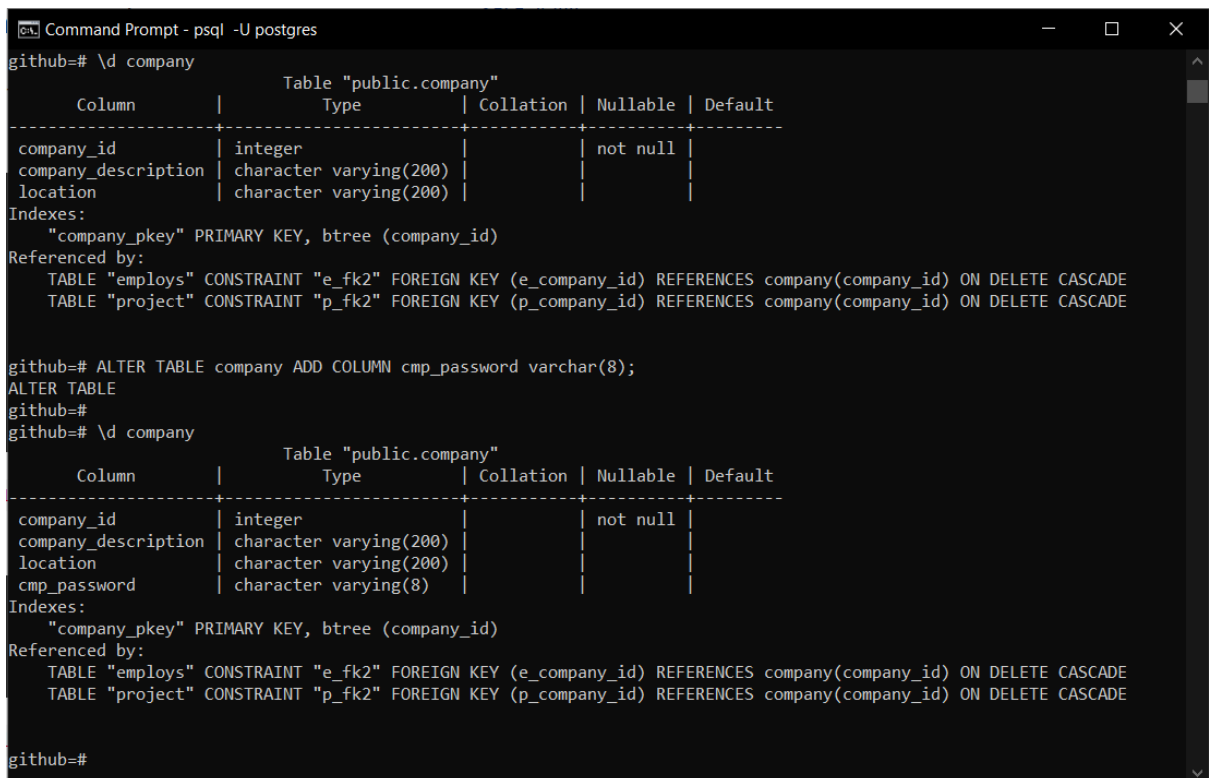
Similarly, the developer has other options such as pulling from the remote repo, pushing the local repo into the remote repo, merging two branches, adding new projects, importing an older version for rollback commits and many more.

The dependencies used for database connectivity are:

- **Python 3** – The front end is implemented as a Command Line Interface(CLI) which is obtained by running a python file. This design choice was taken due to the fact that the actual *git* that we are trying to replicate is also operated using the CLI.
- **Psycopg2** – The [PostgreSQL](#) database adapter for the [Python](#) programming language. This was used to connect the PostgreSQL Database to the frontend running on python.
- **OS** library of Python – This python inbuilt library's functions was used in developing some of the CLI's functionalities.

Change in constraints and schema based on changes in Business/Application changes/expansion:

1. If the Company decides to not provide any developer the responsibility of adding a new user to the repository and want to add it themselves, then can add a `Company_password` column to the "Company" table and use the `Company_id` and Password to login and add the new users to it's projects.



```
Command Prompt - psql -U postgres
github=# \d company
               Table "public.company"
   Column      |      Type      | Collation | Nullable | Default
-----+-----+-----+-----+-----
 company_id    | integer        |           | not null |
 company_description | character varying(200) |           |          |
 location      | character varying(200) |           |          |
Indexes:
    "company_pkey" PRIMARY KEY, btree (company_id)
Referenced by:
    TABLE "employs" CONSTRAINT "e_fk2" FOREIGN KEY (e_company_id) REFERENCES company(company_id) ON DELETE CASCADE
    TABLE "project" CONSTRAINT "p_fk2" FOREIGN KEY (p_company_id) REFERENCES company(company_id) ON DELETE CASCADE

github=# ALTER TABLE company ADD COLUMN cmp_password varchar(8);
ALTER TABLE
github=#
github=# \d company
               Table "public.company"
   Column      |      Type      | Collation | Nullable | Default
-----+-----+-----+-----+-----
 company_id    | integer        |           | not null |
 company_description | character varying(200) |           |          |
 location      | character varying(200) |           |          |
 cmp_password   | character varying(8) |           |          |
Indexes:
    "company_pkey" PRIMARY KEY, btree (company_id)
Referenced by:
    TABLE "employs" CONSTRAINT "e_fk2" FOREIGN KEY (e_company_id) REFERENCES company(company_id) ON DELETE CASCADE
    TABLE "project" CONSTRAINT "p_fk2" FOREIGN KEY (p_company_id) REFERENCES company(company_id) ON DELETE CASCADE

github=#
```

2. Currently inside a version, only different files can be committed, folders cannot be created inside each commit. To implement this file structure, we can add a "parent" column in the files table which provides the folder ID of the folder to which the file belongs to. [the FID can describe both individual files and folders; to discriminate

files from folders, the folders will have their parent column filled with either -1 or the FID of it's parent folder].

```

Command Prompt - psql -U postgres

github=# \d file
               Table "public.file"
   Column   |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 fid        | integer         |           | not null |
 file_name  | character varying(30) |         | not null |
 type       | character varying(10) |         |          | '.txt'::character varying
 file_contains | text           |         |          |
 file_size  | integer         |         | not null |
Indexes:
    "file_pkey" PRIMARY KEY, btree (fid)
Referenced by:
    TABLE "contains" CONSTRAINT "c_fk2" FOREIGN KEY (contains_fid) REFERENCES file(fid)

github=# ALTER TABLE file ADD COLUMN parent_f int;
ALTER TABLE
github=#
github=# \d file
               Table "public.file"
   Column   |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 fid        | integer         |           | not null |
 file_name  | character varying(30) |         | not null |
 type       | character varying(10) |         |          | '.txt'::character varying
 file_contains | text           |         |          |
 file_size  | integer         |         | not null |
 parent_f   | integer         |         |          |
Indexes:
    "file_pkey" PRIMARY KEY, btree (fid)
Referenced by:
    TABLE "contains" CONSTRAINT "c_fk2" FOREIGN KEY (contains_fid) REFERENCES file(fid)

github=#

```

3. If working inside a software product company, there might be a requirement of a issues table which listed all the issues pertaining to a version. This can be envisioned by providing a Issues table/ Relation.

```

Command Prompt - psql -U postgres

github=# CREATE TABLE Issues (
github=# Issue text,
github=# VID int,
github=# V_rem_repo_ID int,
github=# PRIMARY KEY (VID, V_rem_repo_ID));
CREATE TABLE
github=#
github=# ALTER TABLE Issues add constraint i_fk1 FOREIGN KEY (VID,V_rem_repo_ID) REFERENCES Version(VID,V_rem_repo_ID) on delete cascade;
ALTER TABLE
github=#
github=# \d issues
               Table "public.issues"
   Column   |      Type      | Collation | Nullable |      Default
-----+-----+-----+-----+-----
 issue      | text           |           |          |
 vid        | integer         |           | not null |
 v_rem_repo_id | integer         |           | not null |
Indexes:
    "issues_pkey" PRIMARY KEY, btree (vid, v_rem_repo_id)
Foreign-key constraints:
    "i_fk1" FOREIGN KEY (vid, v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id) ON DELETE CASCADE

github=#

```

4. If the developers have hierarchical posts inside a Company, we can add the “post” column to the employs table that describe their post they hold in the company.

```
Command Prompt - psql -U postgres
github=# \d employs
          Table "public.employs"
   Column |      Type      | Collation | Nullable | Default
-----|-----|-----|-----|-----
 e_user_id | integer         |           | not null |
 e_company_id | integer         |           | not null |
  salary   | numeric(10,2)   |           |         |
Indexes:
    "employs_pkey" PRIMARY KEY, btree (e_user_id)
Foreign-key constraints:
    "e_fk1" FOREIGN KEY (e_user_id) REFERENCES developer(user_id) ON DELETE CASCADE
    "e_fk2" FOREIGN KEY (e_company_id) REFERENCES company(company_id) ON DELETE CASCADE

github=# ALTER TABLE employs ADD COLUMN post varchar(20);
ALTER TABLE
github=#
github=# \d employs
          Table "public.employs"
   Column |      Type      | Collation | Nullable | Default
-----|-----|-----|-----|-----
 e_user_id | integer         |           | not null |
 e_company_id | integer         |           | not null |
  salary   | numeric(10,2)   |           |         |
   post    | character varying(20) |           |         |
Indexes:
    "employs_pkey" PRIMARY KEY, btree (e_user_id)
Foreign-key constraints:
    "e_fk1" FOREIGN KEY (e_user_id) REFERENCES developer(user_id) ON DELETE CASCADE
    "e_fk2" FOREIGN KEY (e_company_id) REFERENCES company(company_id) ON DELETE CASCADE

github=#
```

5. Each commit will be done with certain changes implemented in those commits. In order to describe these changes, we can add a “description” column to the Version table in which brief description of the changes done in the commit can be written.

```
Command Prompt - psql -U postgres
github=# \d version
          Table "public.version"
   Column |      Type      | Collation | Nullable | Default
-----|-----|-----|-----|-----
   vid   | integer         |           | not null |
 v_rem_repo_id | integer         |           | not null |
 v_user_id | integer         |           |         |
Indexes:
    "version_pkey" PRIMARY KEY, btree (vid, v_rem_repo_id)
Foreign-key constraints:
    "v_fk1" FOREIGN KEY (v_user_id) REFERENCES developer(user_id) ON DELETE SET NULL
Referenced by:
    TABLE "contains" CONSTRAINT "c_fk1" FOREIGN KEY (contains_vid, contains_v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id)
    TABLE "holds" CONSTRAINT "h_fk2" FOREIGN KEY (vid, v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id)
    TABLE "issues" CONSTRAINT "i_fk1" FOREIGN KEY (vid, v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id) ON DELETE CASCADE

github=# ALTER TABLE version ADD COLUMN description varchar(50);
ALTER TABLE
github=#
github=# \d version
          Table "public.version"
   Column |      Type      | Collation | Nullable | Default
-----|-----|-----|-----|-----
   vid   | integer         |           | not null |
 v_rem_repo_id | integer         |           | not null |
 v_user_id | integer         |           |         |
description | character varying(50) |           |         |
Indexes:
    "version_pkey" PRIMARY KEY, btree (vid, v_rem_repo_id)
Foreign-key constraints:
    "v_fk1" FOREIGN KEY (v_user_id) REFERENCES developer(user_id) ON DELETE SET NULL
Referenced by:
    TABLE "contains" CONSTRAINT "c_fk1" FOREIGN KEY (contains_vid, contains_v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id)
    TABLE "holds" CONSTRAINT "h_fk2" FOREIGN KEY (vid, v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id)
    TABLE "issues" CONSTRAINT "i_fk1" FOREIGN KEY (vid, v_rem_repo_id) REFERENCES version(vid, v_rem_repo_id) ON DELETE CASCADE

github=#
```

Database migration:

In order to increase the scalability, flexibility and performance of the Database Application, the database might be migrated to a NoSQL type database.

If the database needs to be migrated to a NoSQL type database, we recommend migrating to a *wide-Column or column based NoSQL database* such as *Hbase* as it provides both data consistency and scalability.

Hbase also provides versioning of the data with the help of timestamps. This would directly support storing our git's different versions that get committed into the repositories.

The flexibility of not having a pre-existing fixed schema for relations will help in easy handling of complex file-folder system storage and retrieval.

Thank you.