Version Control Database

Nikesh Tadela (22CSB0B21)
Sahil Yadav (22CSB0B20)

March 31, 2024

1 Introduction : What is Version Control ?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It is a crucial aspect of software development that helps teams manage changes to their codebase efficiently. It allows developers to track alterations to their files over time, collaborate effectively, and maintain a clear history of the project's development. Among various version control systems, Git stands out as one of the most popular and powerful tools in this domain.

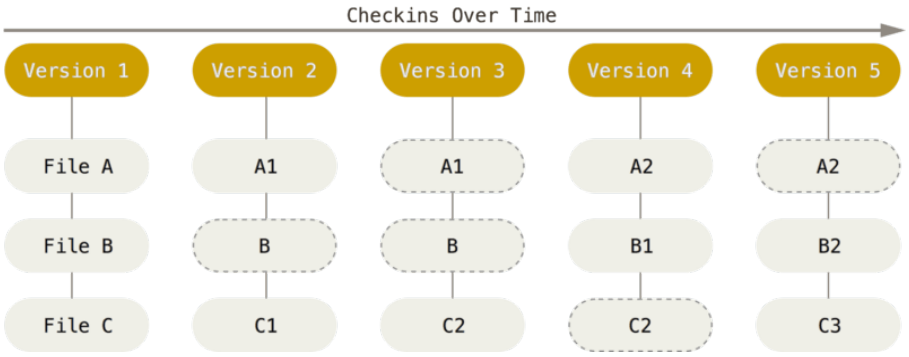

Figure 1: Figure representing how Git stores data

Key Concepts in Git:

1. **Repositories:** In Git, A repository (often abbreviated as "repo") is essentially a data structure that stores all the files, directories, and metadata associated with a project, along with the entire history of changes made to those files.

2. **Commits:** In Git, a commit represents a snapshot of the project at a particular point in time. Each commit records changes to files and includes metadata such as the author's name, email, timestamp, and a unique identifier (SHA-1 hash). Commits serve as checkpoints in the project's history.

3. **Branches:** Branches in Git are lightweight movable pointers to commits. They allow developers to work on isolated features or fixes without affecting the main codebase. Branching enables parallel development and experimentation. The main branch in Git is typically called master or main, but developers can create additional branches as needed.

4. **Merges:** Merging is the process of combining changes from one branch (source branch) into another (target branch). Git performs merges automatically when changes do not conflict. However, if there are conflicting changes (i.e., changes made to the same part of a file in both branches), Git prompts the user to resolve these conflicts manually.

2 Problem Statement

We try to make a Database System for a Version Control Software. The basic responsibilities of our Database System includes :

1. Allow *users* to create *repositories*.
2. Letting contributors to commit to a branch.
3. Allow users to pull out multiple *branches*.
4. Merging a branch with another.
5. Checkout a specific version of the files in the past.
6. See the entire history of a file.
7. Able to represent any folder structure

There are many more responsibilites, These are just to name a few. RDBMS used : PostGresQL.

3 How we store data

3.1 How folder structure is handled ?

As we know, Files are simple entities and just store some data in them. Where as Folders are complicated as they contain files and may also contain other folders too! To tackle this problem, Files and Folders are generalized to an entity called *Item*. A file is an Item and a folder is an Item. Now a file item stores some text/data. A folder item contains other items.

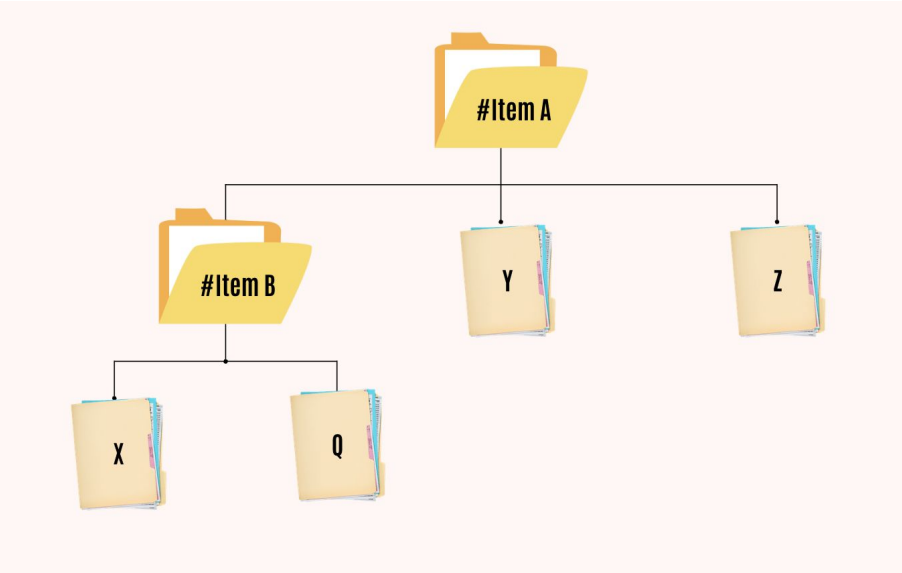

Figure 2: Folder Structure (arrow marks represent contains relation)

3.2 How creations/deletions/changes of a file are handled ?

- **Creation :** A new Item is created. The new commit contains all the existing items in previous commit and also the newly created item.

- **Deletion :** The new commit contains all the existing items in previous commit except for the deleted item.
- **Change :** The new instance of the changed item is created. The new commit is formed by replacing the changed item with new instance of the same item.

3.3 How merges are handled ?

A merge is considered as a special commit to the target branch. So merge is a **specialization** of a commit. It contains extra information such as the source commit and the user who approved the merge.

4 Entity-Relationship Model

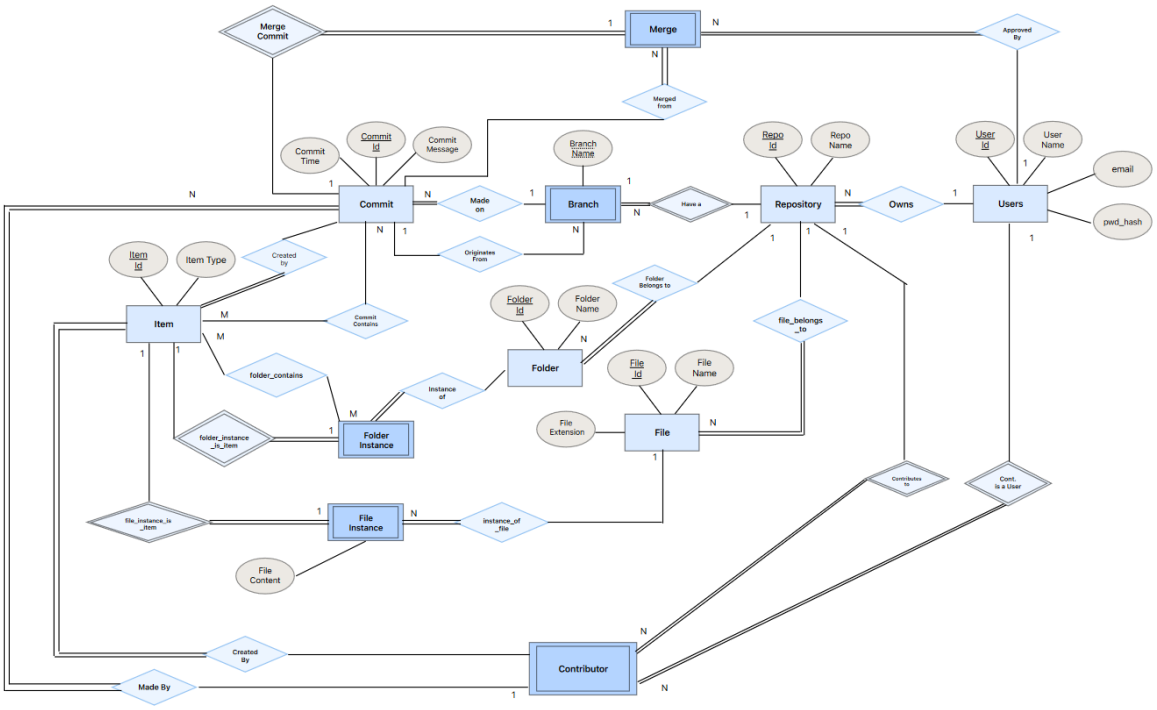

Figure 3: Entity-Relationship Model

Entities

1. **Users** : Represents a user.
   Attributes : user_id ,username, pwd_hash , email
   Primary key : user_id

2. **Repo** : Represents a repository.
   Attributes : repo_id,repo_name

Primary key : repo_id

3. **Contributor(Weak entitiy)** : Represents a contributor to a repo.
   Totally Depends on : users,repo
   Partial Key : No partial key as (user_id,repo_id) uniquely identify a contributor.

4. **Branch (Weak entity)** : Represents a branch to a repo
   Attributes : branch_name
   Parital key : branch_name
   Totally Depends on : repo

5. **Commit** : Represents a commit made to a branch
   Attributes : commit_id,commit_time,commit_message
   Primary key : commit_id

6. **Merge(Weak Entity)** : A special type of commit that merges two branches
   Totally Depends on : commit
   Partial key : No partial key. A commit uniquely identifies a Merge.

7. **Item** : A Generalized entity of a file_instance and a folder_instance
   Attributes : item_id,item_type
   Primary key : item_id

8. **File** : Represents a File in a repo
   Attributes : file_id,file_name,file_extension
   Primary key : file_id

9. **Folder** : Represents a Folder in a repo
   Attributes : folder_id,folder_name
   Primary key : folder_id

10. **file_instance(Weak Entity)** : Represents an instance of a file
    Attributes : file_content
    Totally Depends on : items **Partial key :** An item uniquely identifies a file_instance. No Partial key.

11. **folder_instance(Weak Entity)** : Represents an instance of a folder
    Totally Depends on : items **Partial key :** An item uniquely identifies a folder_instance. No Partial key.

Relationships

1. **Owns**
   For every repo, there must be an owner (who is a user).
   Between : Repo(totally participates) and users
   Assumption : Only a single user owns a repo
   Cardinality : one (user) to many (repo)
   As it is one to many, Primary key of user is added to repo table.

2. **contributor-user**
   Every contributor is a user.
   Between : Contributor(totally depends) and users
   Cardinality : one (user) to many (contributors)
   As it is one to many, Primary key of user is added to contributor table.

3. **contributes-to**
   A contributor must contribute to a repo
   Between : Contributor(totally depends) and repo
   Cardinality : one (repo) to many (contributors)
   As it is one to many, Primary key of repo is added to contributor table.

4. **branch in repo**
   A repo has many branches
   Between : branch(totally depends) and repo
   Cardinality : one (repo) to many (branch)
   Assumption : A branch belongs to a single repo. A repo doesnt contain two branches with same name (as branch name serves as partial key)
   As it is one to many, Primary key of repo is added to branch table.

5. **made on**
   A commit is made on a branch
   Between : commit(totally participates) and branch
   Assumption : A commit is made on a single branch
   Cardinality : one (branch) to many (commits)
   As it is one to many, Primary key of branch is added to commits table.

6. **originates_from**
   A non-main branch originates from a commit
   Between : commits and branch
   Assumption : A branch originates from a single commit
   Cardinality : one (commit) to many (branches)
   As it is one to many, Primary key of commits is added to branches table. Note that, a main branch originates from nothing. So main branches will have null values for that attribute.

7. **made_by**
   A commit must be made by a contributor
   Between : commit(totally participates) and contributor
   Assumption : Only a single user makes a commit
   Cardinality : one (contributor) to many (commits)
   As it is one to many, Primary key of contirbutor is added to commits table.

8. **merge-commit**
   a merge is a special kind of commit
   Between : merge(totally depends) and commit
   Assumption : a commit cannot be multiple merges
   Cardinality : one (commit) to one (merges)
   As it is one to one, Primary key of commits is added to merge table.

9. **merged from**
   A merge must have a source commit.
   Between : merge(totally participates) and commit
   Assumption : There exists only a single source commit
   Cardinality : one (commit) to many (merges)
   As it is one to many, Primary key of commit is added to merge table.

10. **approved by**
    A merge must be approved by a user
    Between : merge(totally participates) and users
    Assumption : Only a single user approves a merge
    Cardinality : one (user) to many (merge)
    As it is one to many, Primary key of user is added to merge table.

11. **commit_contains**
    A commit contains many items
    Between : commits and items
    Assumption : commit contains many items and an item may be contained in many commits.
    Cardinality : many (commit) to many (items)
    As it is many to many, Additional table must be made.

12. **file_instance_is_item**
    Every File instance is an item
    Between : file_instance(totally participates) and file
    Assumption : An item represents a single file instance
    Cardinality : one (item) to one (file_instance)
    As it is one to one, Primary key of item is added to file_instance table.

13. **instance_of_file**
    A file has many file_instances
    Between : file_instance(totally participates) and file
    Assumption : A file will have many file_instance and a file_instance is an instance of a single file
    Cardinality : one (file) to many (file_instance)
    As it is one to many, Primary key of file is added to file_instance table.

14. **File_belongs_to**
    every File belongs to a single repo
    Between : File(totally participates) and Repo
    Assumption : a file belongs to a single repo
    Cardinality : one (repo) to many (file)
    As it is one to many, Primary key of repo is added to file table.

15. **folder_instance_is_item**
    every folder instance is an item
    Between : folder_instance(totally participates) and item
    Assumption : an item represents a single folder instance
    Cardinality : one (item) to one (folder_instance)
    As it is one to one, Primary key of item is added to folder_instance table.

16. **folder_belongs_to**
    every folder belongs to a repo
    Between : folder(totally participates) and repo
    Assumption : a folder belongs to a single repo
    Cardinality : one (repo) to many (folder)
    As it is one to many, Primary key of repo is added to folder table.

17. **folder_contains**
    folder_instance contains many items
    Between : folder_instance and items
    Assumption : folder_instance contains many items and an item can be contained by many folder_instances.
    Cardinality : many (folder_instance) to many (items)
    As it is many to many, A seperate table has to be made.

5 Relational schema and Normalization

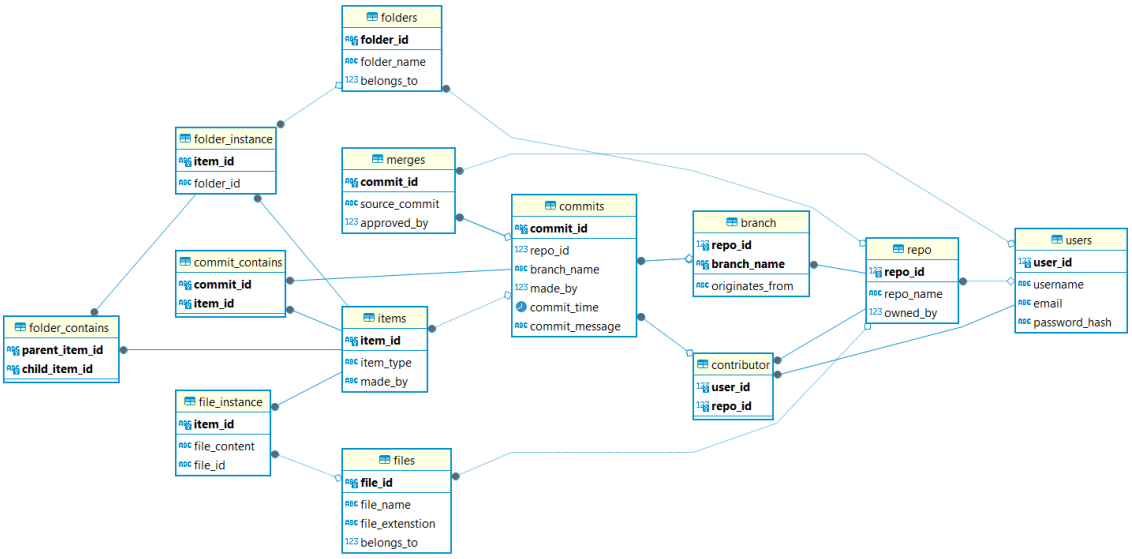

Figure 4: Figure representing the final relation schemas

1. **users**
   FDs : user_id → user_name , user_id →password_hash,user_id → email
   user_id is taken as primary key.
   users relation is in BCNF.

2. **repo**
   FDs : repo_id → repo_name , repo_id →owned_by
   repo_id is taken as primary key.
   repo relation is in BCNF.

3. **contributors**
   FDs : no non trivial FDs
   user_id,repo_id (all the attributes) both together is taken as primary_key.
   users relation is in BCNF.

4. **branch**
   FDs : repo_id,branch_name → originates_from
   repo_id,branch_name both together is taken as primary_key.
   branch relation is in BCNF.

5. **commits**
   FDs : commit_id → commit_message,commit_time,made_by,repo_id,branch_name
   commit_id is taken as primary key.
   commit relation is in BCNF.

6. **merges**
   FDs : commit_id → incoming_commit,approved_by
   commit_id is the primary key.
   merges relation is in BCNF.

7. **item**
   FDs : item_id → item_type, made_by
   item_id is taken as primary key.
   item relation is in BCNF.

8. **file**
   FDs : file_id → file_name,file_extension,belongs_to
   file_id is taken as primary key.
   file relation is in BCNF.

9. **file_instance**
   FDs : item_id → file_id,file_content
   item_id is the primary key.
   file_instance relation is in BCNF.

10. **folder**
    FDs : folder_id → folder_name,belongs_to
    folder_id is taken as primary key.
    folder relation is in BCNF.

11. **folder_instance**
    FDs : item_id → folder_id
    item_id is the primary key.
    folder_instance relation is in BCNF.

12. **folder_contains**
    FDs : No non-trivial FDs
    parent_item_id,child_item_id is the primary key.
    folder_contains relation is in BCNF.

13. **commit_contains**
    FDs : No non-trivial FDs
    commit_id, item_id is taken as the primary key.
    commi_contains relation is in BCNF.

6 SQL queries to create Tables

```
 create table users (
   user_id int primary key,
   username varchar(40) unique,
   email text,
   password_hash text
);

create table repo(
   repo_name varchar(40),
   repo_id int primary key,
   owned_by int,
   foreign key(owned_by) references users
);

create table contributor(
   user_id int,
   repo_id int,
   foreign key (user_id) references users(user_id),
   foreign key(repo_id) references repo(repo_id),
   primary key(user_id,repo_id)
);

create table branch(
   repo_id int references repo(repo_id),
   branch_name varchar(40),
   primary key(repo_id,branch_name)
);
create table merges(
   commit_id varchar(75) references commits(commit_id) primary key,
   incoming_commit varchar(75) references commits(commit_id),
        approved_by int references users(user_id)
);
create table commits(
   commit_id varchar(75) primary key,
   repo_id int,
   branch_name varchar(40),
   made_by int,
   foreign key (repo_id,branch_name) references branch(repo_id,branch_name),
   foreign key (repo_id,made_by) references contributor(repo_id,user_id)
);
create table commit_contains(
   commit_id varchar(75) references commits(commit_id),
   item_id varchar(75) references items(item_id),
```

```
        primary key(commit_id,item_id)
);

create table items(
   item_id varchar(75) primary key,
   item_type varchar(10),
   made_by varchar(75),
   foreign key (made_by) references commits(commit_id)
);

create table files(
   file_id text primary key,
   file_name varchar(40),
   file_extenstion varchar(75),
   belongs_to int,
   foreign key (belongs_to) references repo(repo_id)
);

create table folders(
   folder_id text primary key,
   folder_name varchar(40),
   belongs_to int,
   foreign key (belongs_to) references repo(repo_id)
);

create table file_instance(
   item_id varchar(75) primary key,
   file_content text,
   file_id text,
   foreign key (file_id) references files(file_id),
   foreign key (item_id) references items(item_id)
);

create table folder_instance(
   item_id varchar(75) primary key,
   folder_id text,
   foreign key (item_id) references items(item_id),
   foreign key (folder_id) references folders(folder_id)
);

create table folder_contains(
   parent_item_id varchar(75),
   child_item_id varchar(75),
   foreign key (parent_item_id) references folder_instance(item_id),
   foreign key (child_item_id) references items(item_id),
    primary key(parent_item_id,child_item_id)
);
```

7 SQL queries to insert data

```
INSERT INTO public.users (user_id,username,email,password_hash) VALUES
    (1,'NikeshNikki','nikesh.tadela@gmail.com','asdoif34r0q93i4r0q93iqweif923rqwej');

INSERT INTO public.repo (repo_name,repo_id,owned_by) VALUES
    ('lichesslite',1,1);

INSERT INTO public.contributor (user_id,repo_id) VALUES (1,1);

INSERT INTO public.branch (repo_id,branch_name,originates_from) VALUES
    (1,'main',NULL)
INSERT INTO public.branch (repo_id,branch_name,originates_from) VALUES
    (1,'invitation','119376d1d0ac578bfcc3bffec9bd72855c0cb5a4');

INSERT INTO public.commits
    (commit_id,repo_id,branch_name,made_by,commit_time,commit_message) VALUES
    ('76ef2a5680485ede3da047ee03b4a81110ebf4c6',1,'main',1,'2024-03-20
        19:39:53.375','started out with the boilerplate html code')

INSERT INTO public.merges (commit_id,source_commit,approved_by) VALUES
    ('c1e5ed6e7051e2837e9eb135b2fd206aca386919'
  ,'00f94bb2ae98bbed40cb42d276c947b485542c0b',1);

INSERT INTO public.commit_contains (commit_id,item_id) VALUES
('c1e5ed6e7051e2837e9eb135b2fd206aca386919'
,'f49499a979ea6c4489fb05b53c77e74cb6e7596f04def4cd09efca806e47140e');

INSERT INTO public.file_instance (item_id,file_content,file_id) VALUES
    ('f49499a979ea6c4489fb05b53c77e74cb6e7596f04def4cd09efca806e47140e',
  '<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lichess</title>
</head>
<body>
    <h1>Welcome to Lichess !!</h1>
    <button>Enter the game</button>
    <form action="/login">
        <label for="uname">Username</label>
        <input type="text" id="uname">
        <label for="pwd">Password</label>
        <input type="password" id="pwd">
    </form>
    <p>Hello all !!! You are being invited to participate in this contest</p>
</body>
</html>','1|index.html');

INSERT INTO public.files (file_id,file_name,file_extenstion,belongs_to)
    VALUES
    ('1|index.html','index','html',1);
```

```
INSERT INTO public.folders (folder_id,folder_name,belongs_to) VALUES
    ('2|styles','styles',2);

INSERT INTO public.folder_instance (item_id,folder_id) VALUES
    ('a2982424c72723ef8d2de1a99ec3a138f8633dfee0febd77b8c2e62aa7abfc7a'
  ,'2|styles');

INSERT INTO public.folder_contains (parent_item_id,child_item_id) VALUES
    ('a2982424c72723ef8d2de1a99ec3a138f8633dfee0febd77b8c2e62aa7abfc7a'
  ,'5c81eb00ebfb32f6c8841b12eaffb20d5993d8d5166ebe57b6470ba910f8a8c3');

INSERT INTO public.items (item_id,item_type,made_by) VALUES
    ('373a67541721668c7ba1bcbf884abc56f610eadf9f7073a75e43b88c3f0fbbb6'
  ,'file','76ef2a5680485ede3da047ee03b4a81110ebf4c6');
```

8 Database in Action :

Tech-stack used in a Repo

```
select distinct (file_extenstion) from files where files.belongs_to = 2;
```

Git Log :

```
select cmit.commit_id,users.username,cmit.commit_message,cmit.commit_time
    from
((select * from commits where repo_id = 2 and branch_name = 'main' order by
    commit_time desc) cmit
join users on users.user_id = cmit.made_by);
```

	commit_id	username	commit_message	commit_time
1	cb950126124d0190d5858db25930a2a047bc82ba	Prashanth	deleted first.css	2024-03-17 18:20:32.882
2	8c387ab9031568f3871c7f2af72703af6bd6418b	Prashanth	second.css and top artists heading is made	2024-03-16 18:18:40.918
3	41968ef22811592246907ee022c37c20df240bb5	Sahil	index.html and styles/first.css are made	2024-03-15 18:17:58.169

Figure 5: Git Log

Git log of non-main branch

```
select * from (
(
select * from commits
where branch_name = 'invitation'
and repo_id = 1 order by commit_time desc
)
union
(
select * from commits where (repo_id,branch_name) =
```

```
(select repo_id,branch_name from commits
where commit_id = (select originates_from from branch
                   where branch_name='invitation' and repo_id = 1))
and
commit_time < (select commit_time from commits
               where commit_id = (select originates_from from branch where
                   branch_name='invitation' and repo_id = 1) )
order by commit_time desc
)
);
```

Stats of Contributors

```
select username,coalesce (cmits,0) from
(
    select users.user_id,username from
    (select * from contributor c where c.repo_id = 1) con
    join users on con.user_id = users.user_id
) f
left join
(
    select made_by ,count(*) cmits from commits c where repo_id = 1 group by
        made_by
) s
on f.user_id = s.made_by
;
```

	username	contribution
1	NikeshNikki	3
2	Vinay	2
3	Narayan	0

Figure 6: Contributors Stats

Contents in root in latest commit

```
select folder_id "name",item_id from folder_instance fo where item_id in
(
   select i.item_id from items i where i.item_id in
   (select cc.item_id from commit_contains cc where cc.commit_id =
      (select commit_id from commits where commits.repo_id = 2 order by
          commit_time desc limit 1)
   ) and item_type = 'folder'
)
union
select file_id "name",item_id from file_instance fi where item_id in
```

```
(
   select i.item_id from items i where i.item_id in
   (select cc.item_id from commit_contains cc where cc.commit_id =
      (select commit_id from commits where commits.repo_id = 2 order by
         commit_time desc limit 1)
   ) and item_type = 'file'
);
```

	name	item_id
1	2|index.html	23e897b07040f119b313f44d21d40f057ac930a2ea77ad98f9b4bcd64bd34441
2	2|styles	2bc4bd4ed304a58f50d5d826d4b70403bc643dd604314a63c2a6c3af07a9867c

Figure 7: Contents in root

History of a File

```
select commit_id,commit_message,file_content from
(
select * from commits c where repo_id = 2 and branch_name ='main'
) cmits
join
(
select made_by,file_content from items join
(select * from file_instance
where file_id = '2|index.html') fi on fi.item_id = items.item_id
) fi
on cmits.commit_id = fi.made_by
order by commit_time desc;
```

commit_id	commit_message	file_content	
8c387ab9031568f3871c7f2af72703af6bd6418b	second.css and top artists heading is made	<!DOCTYPE html>¶<html lang="en">¶<head>¶	<meta charset="UTF-8">¶
41968ef22811592246907ee022c37c20df240bb5	index.html and styles/first.css are made	<!DOCTYPE html>¶<html lang="en">¶<head>¶	<meta charset="UTF-8">¶

Figure 8: History of a file