Application description

The database I have chosen to model is that of a Version Control System (VCS) that also has a publicly available API allowing people to query information relating to the database. The database models Users, Repositories, Branches, Commits, Directories, Files, Issues and the relations between these entities. As a version control system is a more flexible file management system it must store info about multiple states of each file and directory. Furthermore to facilitate international usage and special characters the system should support Unicode.

The user is one of the principal entities in a VCS system. A user has a number of personal info fields about them. They are identified by their username which is unique within the system and they also require an associated email address to facilitate contact between the VCS administration and an individual user.

Each user can in turn own a number of repositories. Each repository must be owned by one and only one user. For the purposes of this hypothetical system the only entity which can own repositories are users. Each repository is uniquely identified by a combination of their name and owner (fully qualified name).

Each repo contains a number of branches inside it. Each branch is uniquely identified by an id. Each branch can have multiple users contribute and each user can contribute to multiple branches.

A branch in turn contains a number of directories. It's uniquely identified by its ID. Each directory can belong to multiple branches and multiple branches can have multiple directories. A directory can belong to multiple branches and branches can have multiple directories.

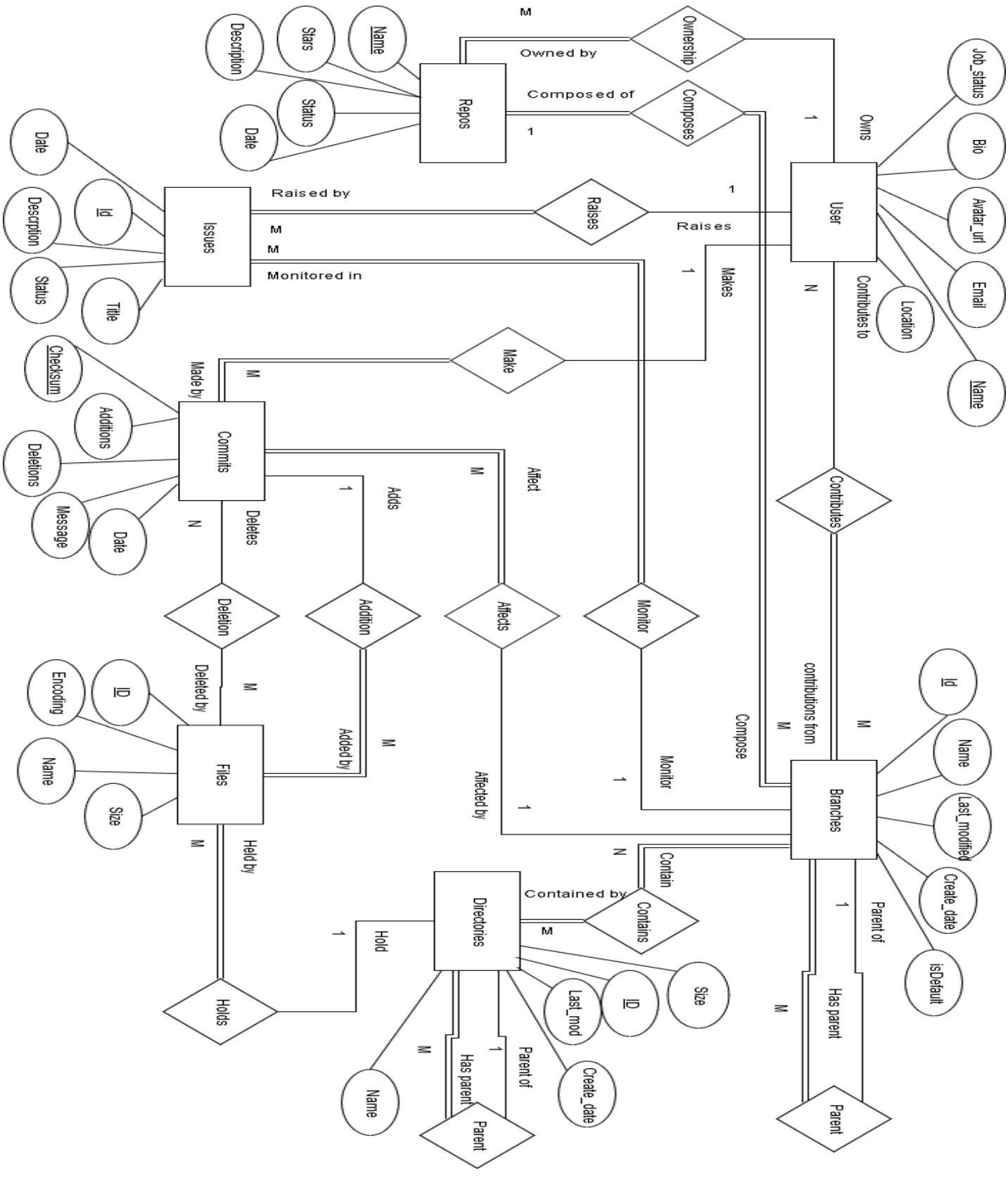
Multiple files can exist inside a directory. Each file can only exist within one repository. Similar to a directory a file is identified by it's identity not it's filename. This is especially important as a commit makes a new instance of a file with the same name. For this exercise the possible modelled encodings are UTF-16 UTF-8 ASCII and Binary with a default of Binary as all files in the end can be represented as Binary.

Commits affect files in two ways, by deleting them (ie. delete or modify) and creating a new one (create or modify). Modifications are modelled as deleting the old instance of the file and creating a new one. A file can be associated with multiple commits if the file exists in different branches. A commit can modify multiple files. A file can only be added by one commit. However it can be deleted by multiple. A commit is uniquely identified by it's checksum.

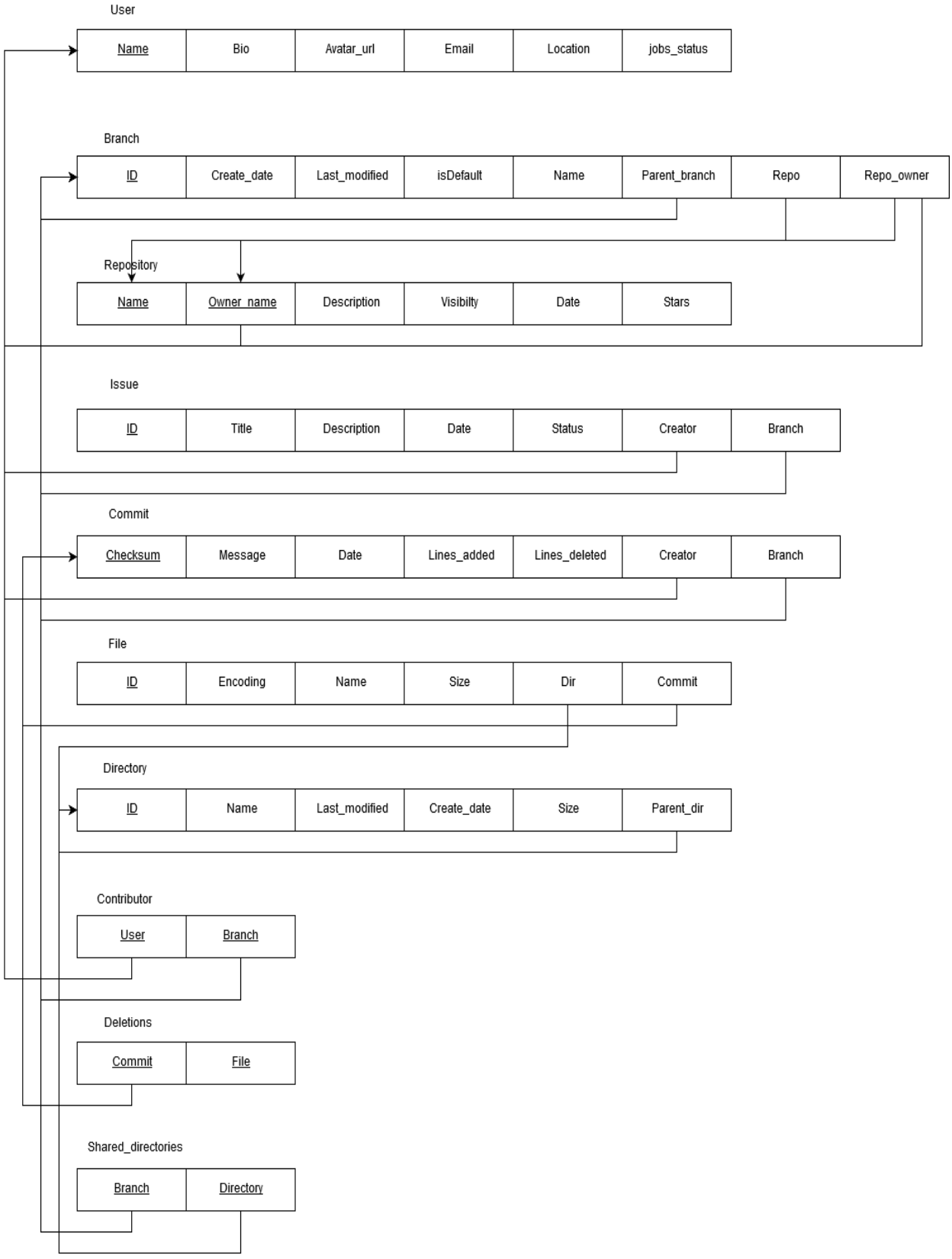
Issues are identified by an unique ID. Each issue can only belong to one branch. If a new branch splits off, then it must create a new instance of the issue.

Idk what extra features are meant to mean. I used variables in the triggers section.

Entity Relationship diagram



Relational Schema



In both diagrams underline indicates primary key. I initially had a creation and last mod date for files but then realised that having it is unnecessary as each file is an individual instance and the creation date is already modelled by the commit.

For normalisation I changed some properties to relationships rather than attributes as I realised, they have a dependency on a primary key ie. parent directory. I split the initial repo name and pathname being modelled as two separate attributes into a composite primary key of owner name and repo name. This was because there was a domain specific dependency that a repo is identified uniquely by a combination of it's own name and the owners name, which formed the pathname.

Tables Definitions:

CREATE TABLE User (Name NVARCHAR(255), Bio NVARCHAR(255), Avatar_url NVARCHAR(255), Email NVARCHAR(255) NOT NULL, Location NVARCHAR(255), Job_status BOOL NOT NULL, PRIMARY KEY(Name));

CREATE TABLE Branch (ID INT NOT NULL, Create_date DATETIME NOT NULL, Last_modified DATETIME NOT NULL, isDefault BOOL NOT NULL, Name NVARCHAR(255) NOT NULL, Parent_branch INT, Repo NVARCHAR(255) NOT NULL, Repo_owner NVARCHAR(255) NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Repo (Name NVARCHAR(255) NOT NULL, Owner NVARCHAR(255) NOT NULL, Description NVARCHAR(255) NOT NULL, Visibility VARCHAR(7) NOT NULL, Date DATETIME NOT NULL, Stars INT NOT NULL, PRIMARY KEY(Name, Owner));

CREATE TABLE Issue (ID INT NOT NULL, Title NVARCHAR(255) NOT NULL, Date DATETIME NOT NULL, Status BOOL NOT NULL, Description NVARCHAR(255) NOT NULL, Creator NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Commit (Checksum INT NOT NULL, Message NVARCHAR(255) NOT NULL, Date DATETIME NOT NULL, Additions INT NOT NULL, Deletions INT NOT NULL, Creator NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(Checksum));

CREATE TABLE File (ID INT NOT NULL, Name NVARCHAR(255) NOT NULL, Encoding VARCHAR(255) NOT NULL, Size INT NOT NULL, Dir INT NOT NULL, Commit INT NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Directory (ID INT NOT NULL, Name NVARCHAR(255) NOT NULL, Size INT NOT NULL, Last_modified DATETIME NOT NULL, Create_date DATETIME NOT NULL, Parent_dir INT, PRIMARY KEY(ID));

CREATE TABLE Contributors (User NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(User, Branch));

CREATE TABLE Deletions (Commit INT NOT NULL, File INT NOT NULL, PRIMARY KEY(Commit, File));

CREATE TABLE Shared_Directories (Branch INT NOT NULL, Dir INT NOT NULL, PRIMARY KEY(Branch, Dir));

Semantic constraints:

- For all tables that have a create date an a last modified date, the last modified date >= create date.
- For commit the lines added and deleted must be >= 0
- The size of all directories and files must be >= 0
- No issue or commit date is < than that of the branch they belong to
- The total number of deletions cannot be greater than the total number of additions

These rules have been achieved using a mixture of triggers and checks

Constraints:

ALTER TABLE User ALTER Job_status SET DEFAULT -1;

ALTER TABLE Issue ADD FOREIGN KEY(Creator) REFERENCES User(Name), ADD FOREIGN KEY(Branch) REFERENCES Branch(ID), ALTER Status SET DEFAULT -1;

ALTER TABLE Commit ADD FOREIGN KEY(Creator) REFERENCES User(Name), ADD FOREIGN KEY(Branch) REFERENCES Branch(ID), ADD CHECK(Additions >= 0 AND Deletions >= 0);

ALTER TABLE Branch ADD FOREIGN KEY(Repo, Repo_owner) REFERENCES Repo(Name, Owner), ADD FOREIGN KEY(Parent_branch) REFERENCES Branch(ID), ADD CHECK (Last_modified >= Create_date);

ALTER TABLE Directory ADD FOREIGN KEY(Parent_dir) REFERENCES Directory(ID),ADD CHECK (Last_modified >= Create_date AND Size >=0);

ALTER TABLE File ADD FOREIGN KEY(Dir) REFERENCES Directory(ID), ADD FOREIGN KEY(Commit) REFERENCES Commit(Checksum), ADD CHECK(Size >= 0 AND Encoding IN (“UTF-8”,”UTF-16”, “Binary”, “ASCII”)), ALTER Encoding SET DEFAULT “Binary”;

ALTER TABLE Repo ADD FOREIGN KEY(Owner) REFERENCES User(Name), ADD CHECK (Visibility IN (“Private”, “Public”)), ALTER Stars SET DEFAULT 0;

ALTER TABLE Contributors ADD FOREIGN KEY(User) REFERENCES User(Name), ADD FOREIGN KEY(Branch) REFERENCES Branch(ID);

ALTER TABLE Deletions ADD FOREIGN KEY(Commit) REFERENCES Commit(Checksum), ADD FOREIGN KEY(File) REFERENCES File(ID);

ALTER TABLE Shared_Directories ADD FOREIGN KEY(Branch) REFERENCES Branch(ID), ADD FOREIGN KEY(Dir) REFERENCES Directory(ID);

Trigger Constraints are located in Trigger section

Security:

- An unidentified person making access to the API to query information about repositories only requires the ability to see public repositories. For this we can create a view of pubic repositories and GRANT SELECT ON Public_repos TO API_User. If the user is found to be abusing this privilege by making too many accesses ti the API (so too many API requests and we want to rate limit them to prevent DDOS/server overload) we can revoke the permission with REVOKE SELECT ON Public_repos FROM API_User. By not propagating the select access with the GRANT OPTION we can be sure we revoked access of public repositories from the user.
- For an authenticated user we want them to be able to see their private repositories and public repositories. We also want them to be able to grant access to others to see their repositories. Thus we can create a new view of all User_Repos. We can then GRANT SELECT ON User_Repos TO Auth_User WITH GRANT OPTION. Again if there are too many hit requests on the user's repos we can revoke this privilege from the user which will propagate through all users with access to see this view. The user themselves can also REVOKE SELECT ON User_Repos FROM Other_User if he decides to block access to his repos from a user they had previously given access to.

Views:

- Public Repos
  CREATE VIEW Public_Repos AS
  SELECT Name, Owner, Stars, Description, Date
  FROM Repo
  WHERE Visibility = "Public";

- User commits
  CREATE VIEW User_Commits AS
  SELECT Name, Commit.Checksum, Date, Message, Additions, Deletions
  FROM User, Commit
  WHERE User.Name = Commit.Creator;

- User repos
  CREATE VIEW User_Repos AS
  SELECT Repo.Name, Repo.Owner, Stars, Description, Date, Visibility

```
FROM Repo, User
WHERE Repo.Owner = “Blue”;
```
*In this case we need to know user Id to create this view. This is an example of making such a view for blue

To see all views in database_name use:
```
SHOW FULL TABLES IN database_name WHERE TABLE_TYPE LIKE 'VIEW';
```

Triggers:

```
delimiter //
CREATE TRIGGER Before_Issue_Insert_Date
BEFORE INSERT ON Issue
FOR EACH ROW
BEGIN
        IF Create_date <
                (SELECT Create_date FROM Branch WHERE Branch.ID = Issue.Branch
                LIMIT 0,1)
        THEN
                SIGNAL SQLSTATE  '45000'
                SET MESSAGE_TEXT = “Issue date < Branch date”;
        END IF;
END;//
delimiter ;
```

“LIMIT 0,1” Limit limits selection to a set size 1 starting at offset 0

Can change above slightly to have checks for all the other date checks as well

```
delimiter //
CREATE TRIGGER Before_Commit_Insert_Owner
BEFORE INSERT ON Commit
FOR EACH ROW
BEGIN
        IF New.Branch NOT IN
                (SELECT Branch FROM Commit)
                AND Creator <>
                (SELECT Repo_owner FROM Branch WHERE Branch.ID = New.Branch
                Limit 0,1)
```

```
            AND (SELECT Parent_branch FROM Branch WHERE Branch.ID =
            New.Branch Limit 0,1) = NULL
        THEN
            SIGNAL SQLSTATE  '45000'
            SET MESSAGE_TEXT = “First commit must belong to repo owner”;
        END IF;
END;//
delimiter ;
```

I really should have had better naming schemes for these triggers…

```
delimiter //
CREATE TRIGGER Before_Commit_Insert_Sum
BEFORE INSERT ON COMMIT
FOR EACH ROW
BEGIN
        DECLARE adds INT;
        DECLARE dels INT;
        SET adds = (SELECT SUM(Additions) FROM Commit) + New.Additions;
        SET dels = (SELECT SUM(Deletions) FROM Commit) + New.Deletions;
        IF  dels > adds
        THEN
            SIGNAL SQLSTATE VALUE '45000'
            SET MESSAGE_TEXT = “More total additions than deletions”;
        END IF;
END;//
delimiter ;
```

Update Operations, Relational Selects and Joins:

User Blue changes job status to looking for a job

```
UPDATE User
SET Job_status =0
WHERE Name = “Blue”;
```

User Blue relocates to space

```
UPDATE User
SET Location = “Space”
WHERE Name = “Blue”;
```

Plenty of Relational selects and joins used in view creation and inside some of the triggers.

More updates would be trivial so not going to bother giving more examples

Sample database Appendix:

//Note that mysql has poor support for Booleans and does not properly adhere to the SQL Bool standards currently and Bool is just an alias for tinyint where any non-zero value indicates false. Also note that I didn't add any of the constraint checks until after I entered in the data as suggested in one of the lectures because constraint checks make it so you have to enter in data in a very specific order.

CREATE TABLE User (Name NVARCHAR(255), Bio NVARCHAR(255), Avatar_url NVARCHAR(255), Email NVARCHAR(255) NOT NULL, Location NVARCHAR(255), Job_status BOOL NOT NULL, PRIMARY KEY(Name));

CREATE TABLE Branch (ID INT NOT NULL, Create_date DATETIME NOT NULL, Last_modified DATETIME NOT NULL, isDefault BOOL NOT NULL, Name NVARCHAR(255) NOT NULL, Parent_branch INT, Repo NVARCHAR(255) NOT NULL, Repo_owner NVARCHAR(255) NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Repo (Name NVARCHAR(255) NOT NULL, Owner NVARCHAR(255) NOT NULL, Description NVARCHAR(255) NOT NULL, Visibility VARCHAR(7) NOT NULL, Date DATETIME NOT NULL, Stars INT NOT NULL, PRIMARY KEY(Name, Owner));

CREATE TABLE Issue (ID INT NOT NULL, Title NVARCHAR(255) NOT NULL, Date DATETIME NOT NULL, Status BOOL NOT NULL, Description NVARCHAR(255) NOT NULL, Creator NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Commit (Checksum INT NOT NULL, Message NVARCHAR(255) NOT NULL, Date DATETIME NOT NULL, Additions INT NOT NULL, Deletions INT NOT NULL, Creator NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(Checksum));

CREATE TABLE File (ID INT NOT NULL, Name NVARCHAR(255) NOT NULL, Encoding VARCHAR(255) NOT NULL, Size INT NOT NULL, Dir INT NOT NULL, Commit INT NOT NULL, PRIMARY KEY(ID));

CREATE TABLE Directory (ID INT NOT NULL, Name NVARCHAR(255) NOT NULL, Size INT NOT NULL, Last_modified DATETIME NOT NULL, Create_date DATETIME NOT NULL, Parent_dir INT, PRIMARY KEY(ID));

CREATE TABLE Contributors (User NVARCHAR(255) NOT NULL, Branch INT NOT NULL, PRIMARY KEY(User, Branch));

```
CREATE TABLE Deletions (Commit INT NOT NULL, File INT NOT NULL, PRIMARY
KEY(Commit, File));

CREATE TABLE Shared_Directories (Branch INT NOT NULL, Dir INT NOT NULL, PRIMARY
KEY(Branch, Dir));

ALTER TABLE User ALTER Job_status SET DEFAULT -1;

ALTER TABLE Issue ADD FOREIGN KEY(Creator) REFERENCES User(Name), ADD FOREIGN
KEY(Branch) REFERENCES Branch(ID), ALTER Status SET DEFAULT -1;

ALTER TABLE Commit ADD FOREIGN KEY(Creator) REFERENCES User(Name), ADD
FOREIGN KEY(Branch) REFERENCES Branch(ID), ADD CHECK(Additions >= 0  AND
Deletions >= 0);

ALTER TABLE Branch ADD FOREIGN KEY(Repo, Repo_owner) REFERENCES Repo(Name,
Owner), ADD FOREIGN KEY(Parent_branch) REFERENCES Branch(ID), ADD CHECK
(Last_modified >= Create_date);

ALTER TABLE Directory ADD FOREIGN KEY(Parent_dir) REFERENCES Directory(ID),ADD
CHECK (Last_modified >= Create_date AND Size >=0);

ALTER TABLE File ADD FOREIGN KEY(Dir) REFERENCES Directory(ID), ADD FOREIGN
KEY(Commit) REFERENCES Commit(Checksum), ADD CHECK(Size >= 0 AND Encoding IN
("UTF-8","UTF-16", "Binary", "ASCII")), ALTER Encoding SET DEFAULT "Binary";

ALTER TABLE Repo ADD FOREIGN KEY(Owner) REFERENCES User(Name), ADD CHECK
(Visibility IN ("Private", "Public")), ALTER Stars SET DEFAULT 0;

ALTER TABLE Contributors ADD FOREIGN KEY(User) REFERENCES User(Name), ADD
FOREIGN KEY(Branch) REFERENCES Branch(ID);

ALTER TABLE Deletions ADD FOREIGN KEY(Commit) REFERENCES Commit(Checksum),
ADD FOREIGN KEY(File) REFERENCES File(ID);

ALTER TABLE Shared_Directories ADD FOREIGN KEY(Branch) REFERENCES Branch(ID),
ADD FOREIGN KEY(Dir) REFERENCES Directory(ID);

INSERT INTO User VALUES("Blue", "A blue person", "defaultpic03.jpg",
"Blue@gmail.com", "Ireland" , -1);
```

```
INSERT INTO User VALUES(“Smiley”, “A smiley person”, “defaultpic01.jpg”,
“Smiley@gmail.com”, “Britain” , -1);
INSERT INTO User VALUES(“Monster”, “A monster person”, “defaultpic02.jpg”,
“Monster@gmail.com”, “USA” , -1);
INSERT INTO User VALUES(“Person”, “A person”, “defaultpic04.jpg”,
“Person@gmail.com”, “China” , -1);
INSERT INTO User VALUES(“Guest”, “A guest person”, “defaultpic05.png”,
“Guest@gmail.com”, NULL , 0);

INSERT INTO Repo VALUES(“Prolog”, “Blue”, “A practise repo for Prolog”, “Public” ,
“2018-01-01 12:00:00”, 0);
INSERT INTO Repo VALUES(“Prolog”, “Monster”, “Prolog repo”, “Public” , “2018-01-01
12:00:00”, 0);
INSERT INTO Repo VALUES(“Haskell”, “Guest”, “A practise repo for Haskell”, “Private” ,
“2017-01-01 12:59:00”, 0);
INSERT INTO Repo VALUES(“ARM”, “Smiley”, “A practise repo for Arm”, “Public” , “2018-
01-01 12:00:00”, 0);
INSERT INTO Repo VALUES(“Javascript”, “Blue”, “A practise repo for JS”, “Public” ,
“2018-01-01 12:00:00”, 0);
INSERT INTO Repo VALUES(“Python”, “Blue”, “A practise repo for Python”, “Private” ,
“2018-01-01 12:00:00”, 0);
INSERT INTO Repo VALUES(“C”, “Smiley”, “C repo”, “Public” , “2018-01-01 12:00:00”, 7);

INSERT INTO Branch VALUES(0, “2018-01-01 12:00:00”, “2018-01-02 12:00:00”, 0,
“Master”, NULL, “Prolog”, “Blue”);
INSERT INTO Branch VALUES(1, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, 0,
“Master”, NULL, “Prolog”, “Monster”);
INSERT INTO Branch VALUES(2, “2017-01-01 12:59:00” , “2017-01-01 12:59:00”, 0,
“Master”, NULL, “Haskell”, “Guest”);
INSERT INTO Branch VALUES(3, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, 0,
“Master”, NULL, “ARM”, “Smiley”);
INSERT INTO Branch VALUES(4, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, 0,
“Master”, NULL, “Javascript”, “Blue”);
INSERT INTO Branch VALUES(5, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, 0,
“Master”, NULL, “Python”, “Blue”);
INSERT INTO Branch VALUES(6, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, 0,
“Master”, NULL, “C”, “Smiley”);
INSERT INTO Branch VALUES(7, “2018-01-02 12:00:00”, “2018-01-02 12:00:00”, -1,
“Potato”, 0, “Prolog”, “Blue”);
```

```
INSERT INTO Issue VALUES(0,”Unstable on a potato”, “2018-01-01 12:00:00”, 0, “My potato crashes when running ur code”, “Guest”, 0);
INSERT INTO Issue VALUES(1,”Unstable on a potato”, “2018-01-02 12:00:00”, -1, “My potato crashes when running ur code”, “Guest”, 7);


INSERT INTO Commit VALUES(0,”Added Readme”,”2018-01-01 12:00:00”, 100, 0, “Blue”, 0);
INSERT INTO Commit VALUES(1,”Works on certain species of tubers now”,”2018-01-02 12:00:00”, 10, 20, “Monster”, 0);
INSERT INTO Commit VALUES(2,”Works now”,”2018-01-02 12:00:00”, 10, 20, “Blue”, 7);
INSERT INTO Commit VALUES(3,”Init repo”,”2018-01-02 12:00:00”, 10, 0, “Monster”, 1);
INSERT INTO Commit VALUES(4,” Init repo”, “2017-01-01 12:59:00”  , 10, 0, “Guest”, 2);
INSERT INTO Commit VALUES(5,” Init repo”,”2018-01-02 12:00:00”, 10, 0, “Smiley”, 3);
INSERT INTO Commit VALUES(6,” Init repo”,”2018-01-02 12:00:00”, 10, 0, “Blue”, 4);
INSERT INTO Commit VALUES(7,” Init repo”,”2018-01-02 12:00:00”, 10, 0, “Blue”, 5);
INSERT INTO Commit VALUES(8,” Init repo”,”2018-01-02 12:00:00”, 10, 0, “Smiley”, 6);


INSERT INTO File VALUES(0,”Readme.md”, “UTF-8”, 1000, 0, 0);
INSERT INTO File VALUES(1,”Readme.md”, “UTF-8”, 1000, 0, 1);
INSERT INTO File VALUES(2,”Readme.md”, “UTF-8”, 1000, 0, 2);
INSERT INTO File VALUES(3,”Readme.md”, “UTF-8”, 1000, 1, 3);
INSERT INTO File VALUES(4,”Readme.md”, “UTF-8”, 1000, 2, 4);
INSERT INTO File VALUES(5,”Readme.md”, “UTF-8”, 1000, 3, 5);
INSERT INTO File VALUES(6,”Readme.md”, “UTF-8”, 1000, 4, 6);
INSERT INTO File VALUES(7,”Readme.md”, “UTF-8”, 1000, 5, 7);
INSERT INTO File VALUES(8,”Readme.md”, “UTF-8”, 1000, 6, 8);


INSERT INTO Directory VALUES( 0, “root”, 1001, “2018-01-02 12:00:00”, “2018-01-01 12:00:00”, NULL);
INSERT INTO Directory VALUES( 1, “root”, 1001, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, NULL);
INSERT INTO Directory VALUES( 2, “root”, 1001, “2017-01-01 12:59:00”, “2017-01-01 12:59:00”, NULL);
INSERT INTO Directory VALUES( 3, “root”, 1001, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, NULL);
INSERT INTO Directory VALUES( 4, “root”, 1001, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, NULL);
```

```
INSERT INTO Directory VALUES( 5, “root”, 1001, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, NULL);
INSERT INTO Directory VALUES( 6, “root”, 1001, “2018-01-01 12:00:00”, “2018-01-01 12:00:00”, NULL);
INSERT INTO Directory VALUES( 7, “tuber”, 1, “2018-01-02 12:00:00”, “2018-01-02 12:00:00”, 0);

INSERT INTO Contributors VALUES(“Blue”, 0);
INSERT INTO Contributors VALUES(“Monster”, 0);
INSERT INTO Contributors VALUES(“Monster”, 1);
INSERT INTO Contributors VALUES(“Guest”, 2);
INSERT INTO Contributors VALUES(“Smiley”, 3);
INSERT INTO Contributors VALUES(“Blue”, 4);
INSERT INTO Contributors VALUES(“Blue”, 5);
INSERT INTO Contributors VALUES(“Smiley”, 6);
INSERT INTO Contributors VALUES(“Blue”, 7);

INSERT INTO Deletions VALUES(1,0);
INSERT INTO Deletions VALUES(2,0);

INSERT INTO Shared_Directories VALUES(0,0);
INSERT INTO Shared_Directories VALUES(1,1);
INSERT INTO Shared_Directories VALUES(2,2);
INSERT INTO Shared_Directories VALUES(3,3);
INSERT INTO Shared_Directories VALUES(4,4);
INSERT INTO Shared_Directories VALUES(5,5);
INSERT INTO Shared_Directories VALUES(6,6);
INSERT INTO Shared_Directories VALUES(7,0);
```

I basically make a bunch of user and repos which is straightforward enough. All of these repos have an associated root branch and each root branch has an associated root directory. The confusing aspect is after that I basically give a divergent branch in the Blue/Prolog repo. The contributors to that project are Blue and Monster with blue making a commit in both branches and monster in only the parent branch. Both branches have a commit which are pretty similar and change the same file in a similar manner. Also to add to that Guest raises an issue in the parent branch which is marked resolved in the child branch but not in the parent. Also the parent branch gets a new nested directory called tubers which is not shared with the child directory. Pretty much every repo gets an initialisation commit and a readme

Kind of snarky comment – any time you think “wait a minute can’t I deduce this from these two non-primary key attributes” the answer is no, in my system you can’t, if it’s not a primary key then you can have duplicates of that column. Remember it depends on the key, the whole key and nothing but the key. So for example you’re allowed to have two issues with the same name in the same branch no problem as long as they have different IDs.