**CPSC 304 Project Cover Page**

Milestone #: 2

Date: 2024-07-12

Group Number: 18

| **Name** | **Student Number** | **CS Alias (Userid)** | **Preferred E-mail Address** |
| --- | --- | --- | --- |
| Charity Grey | 81808313 | s1i9f | charity.grey1@gmail.com |
| Marcus Guay | 57747115 | u2y6v | marcus.guay.99@gmail.com |
| Sarah Yu | 77021384 | l0t3a | sarahjxyu07@gmail.com |

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

## 

## **2Deliverables**

Each group must provide the following as a single PDF file:

1. A completed cover page (template on Canvas) Done
2. **A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.**

We wish to make a version control system like Github. Users will be able to add, delete and edit files and folders, and commit those actions to a repository on any branch they would like, as well as the ability to revert those changes if necessary.

1. **The ER diagram you are basing your item #3 (below) on.** This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.

We have made changes to our ER diagram in the following ways:

* Added entity Issues: to fulfill requirement of 7 entities
* Added non-key attributes in the entities: Folders, files and Branches to fulfill requirement that each Entity must have one non-key attribute
* Added ISA constraints Partial + Disjoint (previously missing)
* Added branchedFrom relationship from Branch to Commit to encapsulate previously known information
* Move comments on commits to comments on Issues

|  |
| --- |

1. **The schema derived from your ER diagram (above).** For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:
2. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.

| **Potential Schema**  **Entity Tables:**  Users(id: INT, hashPassword: VARCHAR(50), username: VARCHAR(30), dateJoined: DATE, email: VARCHAR(320)  Repo( id: INT, name: VARCHAR(50), dateCreated: DATE)  Issues( id: INT, description : VARCHAR(50), dateResolved: DATE, **repoID: int**)  Note: renamed userid and issueid for differentiation  Comments(id: INT, **userid: INT, issueId: INT,** message: VARCHAR(1000), timePosted : TIME)  Note: renamed userid and issueid for differentiation  Branch( **repoid: INT**, **fromCommitID: INT,** name: VARCHAR(100), createdOn: DATE)  Note: name should be underlined with a dotted line; repoid was renamed for clarification  Commits(id: INT, dateCreated: DATE, message: VARCHAR(250), **branchName: varchar(50), repoid: INT, creatorUserID: INT)**  Note: renamed branchName and repoid and creatorUserID for differentiation  Files(Id: INT , path: VARCHAR(4096), createdOn: DATE, **blobHash: VARCHAR(64)**)  Folders(**id: INT**, numberOfFiles: INT)  Blob(hash: VARCHAR(64), contents: VARCHAR(65,535))  **Relationship Tables:**  IssuesAssignedTo(**userid: INT**, **issueId: INT**)  UserContributesTo(**userId: INT**, **repoID: INT**, permissions: INT, edit: boolean, owner:boolean)  Note: renamed userid and repoid for differentiation  CommitsAndFolders(**FolderID: INT, commitId: INT** )  Note: renamed FolderID and commitId for differentiation of id attribute  FilesInFolders(**FolderID: INT**, **FileID: INT**) |
| --- |

1. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.

**Primary Key:**

Files: FilePath

Users: UserID

Comments: CommentID

Issues: IssueID

Repo: RepoID

Commit: CommitID

Blob: BlobHash

Branch: RepoID+ BranchName

IssuesAssignedTo: UserID and IssueID

UserContributesTo: UserId and RepoID

CommitsAndFolders: path and CommitID

**Candidate Key:**

User: Username

Repo: Name

Blob: contents

**Foreign Keys:**

Comments: UserID and IssueID

Issues: RepoID

Commits: BranchName, repoID and creatorUserId

Branch: RepoID, fromCommitID

Files: blobHash

Folders: Path

IssuesAssignedTo: UserID and IssueID

UsersContributesTo: UserId and RepoId

CommitsAndFolders: Path and CommitID

**Constraints:**

**NOT NULL** Email in User

**NOT NULL** repoID in Issues

**NOT NULL** userID and issueID in Comments

**NOT NULL** fromCommitID in Branch

**NOT NULL** RepoID in Branch

**NOT NULL** BranchName, repoID and creatorUserId in Commits

**UNIQUE NOT NULL** Username in User

**UNIQUE NOT NULL** name in Repo

**NOT NULL** Path in Files

NOT NULL BlobHash in Files

1. **Functional Dependencies (FDs)**
2. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key). PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

* Email determines username for users? Or maybe email determines id. Can one email make multiple accounts?
* User ID + repo can determine permissions, if that counts?
* From the file path we can determine the folder, and that folder’s folder, and etc
* Commit ID determines message
* Content determines the Hash for a Blob

**Primary Key Dependencies (Based on definition of primary key):**

Files

* FileID–> CreatedOn

Folders

* FileID –> numberOfFiles

Users

* ID -> hashPassword, username, User dateJoined, User Email

Comments

* ID -> Message and timePosted

Issues:

* ID -> dateResolved, Issue ID, Issue description

Repo

* ID -> Repo name, Repo Date Created

Commit

* ID -> Commit Date created, Commit message

Blob:

* Hash -> Blob contents

Branch

* RepoID + BranchName -> CreatedOn, fromCommitID

UserContributesTo:

* UserId and RepoID -> permissions

**Other:**

Blob

* Content -> Hash (Hash is based on file content through the hashing algorithm)

Users

* UserName -> ID
* Email -> HashPassword

Repo

* Name -> ID

Note: In your list of FDs, there must be some kind of valid FD other those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise.

1. **Normalization**
2. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization. You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, ...). ALL Tables must be listed, not only the ones post normalization.

| Dependencies :  Left hand side is primary key dependencies:   * FileID –> CreatedOn * FileID–> numberOfFiles * UserID -> hashPassword, username, User dateJoined, User Email * CommentID -> Message and timePosted * IssueID -> dateResolved, Issue ID, Issue description * RepoID -> Repo name, Repo Date Created * CommitID -> Commit Date created, Commit message * FileHash -> Blob contents * RepoID + BranchName -> CreatedOn, fromCommitID * UserId and RepoID -> permissions   Left hand side is candidate key dependencies:   * BlobContent -> BlobHash * RepoName ->RepoID * UserName -> UserID   ONLY these could violate BCNF since the rest depend on primary keys/candidate keys   * Email-> HashPassword * Permissions -> edit * Permissions -> owner   Notes : Added entity name in front of attributes to distinguish them |
| --- |
| **Actual Normalization:**  Users(id: INT, hashPassword: VARCHAR(50), username: VARCHAR(30), dateJoined: DATE, email: VARCHAR(320))  Relevant FDs:   * Email-> HashPassword violates BCNF because Email is not a superkey or a candidate key.   We can decompose the relation as:  Users1(email: VARCHAR(320), hashPassword: VARCHAR(50)) IN BCNF and  Users2(id: INT, username: VARCHAR(30), dateJoined: DATE, email: VARCHAR(320)) IN BCNF  Explanation: There are no relevant FDs remaining in Users2 because all FDs related to this are either a primary key or a candidate key on the left hand side.  Users1 is in BCNF due to being a two attribute entity.  Repo( id: INT, name: VARCHAR(50), dateCreated: DATE) IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Issues( id: INT, description : VARCHAR(50), dateResolved: DATE, **repoID: int**) IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Comments(id: INT, **userid: INT, issueId: INT,** message: VARCHAR(1000), timePosted : TIME) IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Branch( **repoid: INT**, **fromCommitID: INT,** name: VARCHAR(100), createdOn: DATE) IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Commits(id: INT, dateCreated: DATE, message: VARCHAR(250), **branchName: varchar(50), repoid :INT, creatorUserID: INT)** IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Files(Id: INT , path: VARCHAR(4096), createdOn: DATE, **blobHash: VARCHAR(64)**) IN BCNF  Explanation: There are no relevant FDs remaining because all FDs related to this are either a primary key or a candidate key on the left hand side.  Folders(**id: INT**, numberOfFiles: INT) IN BCNF  Explanation: 2 attribute relations are in BCNF  Blob(hash: VARCHAR(64), contents: VARCHAR(65,535)) IN BCNF  Explanation: 2 attribute relations are in BCNF  **Relationship Tables:**  IssuesAssignedTo(**userid: INT**, **issueId: INT**) IN BCNF  Explanation: 2 attribute relations are in BCNF  UserContributesTo(**userId: INT**, **repoID: INT**, permissions INT, edit:boolean, owner:boolean)  The following FD are violated.  Permissions -> edit, owner  We will decompose this table:  Permissions(permissions INT, edit:boolean, owner: boolean)  UserContributesTo(**userId: INT**, **repoID: INT**, **permissions INT**)  Both tables are now in BCNF  CommitsAndFolders(**FolderID: INT, commitId: INT** ) IN BCNF  Explanation: 2 attribute relations are in BCNF  FilesInFolders(**FolderID: INT, FileID: INT**) IN BCNF  Explanation: 2 attribute relations are in BCNF |

1. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc. Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

| Table | DDL Statement   * All tested with <https://sqlfiddle.com/> Oracle |
| --- | --- |
| Users1 | CREATE TABLE Users1 (  email VARCHAR(320),  hashPassword: VARCHAR(50),  PRIMARY KEY (email)  ); |
| Users2 | CREATE TABLE Users2(  id INT,  username VARCHAR(30) UNIQUE NOT NULL,  dateJoined DATE,  email VARCHAR(320) NOT NULL,  PRIMARY KEY (id),  FOREIGN KEY (email)  REFERENCES User1(email)  ); |
| Repo | CREATE TABLE Repo(  id INT,  name VARCHAR(50) UNIQUE NOT NULL,  dateCreated: DATE,  PRIMARY KEY (id)  ); |
| Issues | CREATE TABLE Issues(  id INT,  description VARCHAR(50),  dateResolved DATE,  repoID INT NOT NULL,  PRIMARY KEY (id),  FOREIGN KEY (repoID)  REFERENCES Repo(id)  ) |
| Comments | CREATE TABLE Comments(  id INT,  userid INT NOT NULL,  issueId INT NOT NULL,  message VARCHAR(1000),  timePosted TIME,  PRIMARY KEY (id),  FOREIGN KEY (userid)  REFERENCES Users2(id),  FOREIGN KEY (issueId)  REFERENCES Issues(id)  ); |
| Branch | CREATE TABLE Branch(  repoid INT NOT NULL,  name VARCHAR(100),  fromCommitID INT,  createdOn DATE,  PRIMARY KEY (repoid, name),  FOREIGN KEY (repoid)  REFERENCES Repo(id),  FOREIGN KEY (fromCommitID)  REFERENCES Commit(id)  ); |
| Commits | CREATE TABLE Commits(  id: INT,  dateCreated: DATE,  message: VARCHAR(250),  repoid: INT NOT NULL,  branchName: varchar(50) NOT NULL,  creatorUserID: INT NOT NULL,  PRIMARY KEY (id),  FOREIGN KEY (repoid, branchName)  REFERENCES Branch(repoid, name),  FOREIGN KEY (creatorUserID)  REFERENCES Users2(id),  ); |
| Blob | CREATE TABLE Blob(  hash: VARCHAR(64),  content: VARCHAR(65535)  PRIMARY KEY (hash)  ) |
| Files | CREATE TABLES Files(  id INT,  path VARCHAR(4096) NOT NULL,  createdOn DATE,  blobHash VARCHAR(64) NOT NULL,  PRIMARY KEY (id),  FOREIGN KEY (blobHash)  REFERENCES Blob(blobHash)  ); |
| Folders | CREATE TABLES Folders(  id INT,  numberOfFiles INT,  PRIMARY KEY (id),  FOREIGN KEY (id)  REFERENCES Files(id)  ); |
| IssuesAssignedTo | CREATE TABLE IssuesAssignedTo(  userid INT,  issueid INT,  PRIMARY KEY (userid, issueid),  FOREIGN KEY (userid)  REFERENCES Users2(id),  FOREIGN KEY (issueid)  REFERENCES Issues(id)  ); |
| UserContributesTo | CREATE REPO UserContributesTo (  userid INT,  repoid INT,  permissions INT,  PRIMARY KEY (userid, repoid),  FOREIGN KEY (userid)  REFERENCES Users2(id),  FOREIGN KEY (repoid)  REFERENCES Repo(id),  FOREIGN KEY (permissions)  REFERENCES Permissions(permissions)  ); |
| Permissions | CREATE REPO Permissions(  permissions INT,  edit BOOLEAN NOT NULL,  owner BOOLEAN NOT NULL  PRIMARY KEY (permissions)  ); |
| CommitsAndFolders | CREATE REPO CommitsAndFolders(  folderId INT,  commitId INT,  PRIMARY KEY (folderId , commitId),  FOREIGN KEY (folderId)  REFERENCES Folders(id),  FOREIGN KEY (commitId)  REFERENCES Commits(id)  ); |
| FilesInFolders | CREATE FilesInFolders(  FolderID: INT,  FileID: INT,  PRIMARY KEY (FolderID , FileID),  FOREIGN KEY (folderId)  REFERENCES Folders(id),  FOREIGN KEY (FileID)  REFERENCES Files(id)  ) |

1. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later.

| Table of insert | Insert Query |
| --- | --- |
| Users1 | INSERT INTO Users1(email, hashPassword) VALUES ('test@gmail.com', '12345678910');  INSERT INTO Users1(email, hashPassword) VALUES ('sheep@gmail.com', '3247104203914870');  INSERT INTO Users1(email, hashPassword) VALUES ('cow@gmail.com', '987041142');  INSERT INTO Users1(email, hashPassword) VALUES ('cat@gmail.com', '904872135');  INSERT INTO Users1(email, hashPassword) VALUES ('mouse@gmail.com', '701497098'); |
| Users2 | INSERT INTO Users2(id,username,dateJoined, email) VALUES (  1, 'test\_account', TO\_DATE('2024/07/03', 'yyyy/mm/dd'), 'test@gmail.com'  );  INSERT INTO Users2(id,username,dateJoined, email) VALUES (  2, 'iamasheep', TO\_DATE('2024/07/04', 'yyyy/mm/dd'), 'sheep@gmail.com'  );  INSERT INTO Users2(id,username,dateJoined, email) VALUES (  3, 'old\_mcdonald', TO\_DATE('2024/07/05', 'yyyy/mm/dd'), 'cow@gmail.com'  );  INSERT INTO Users2(id,username,dateJoined, email) VALUES (  4, 'cat\_account1', TO\_DATE('2024/07/06', 'yyyy/mm/dd'), 'cat@gmail.com'  );  INSERT INTO Users2(id,username,dateJoined, email) VALUES (  5, 'cat\_account2', TO\_DATE('2024/07/18', 'yyyy/mm/dd'), 'cat@gmail.com'  ); |
| Repo | INSERT INTO Repo(id,name,dateCreated) VALUES (  1, 'test\_repository', TO\_DATE('2024/07/08', 'yyyy/mm/dd')  );  INSERT INTO Repo(id,name,dateCreated) VALUES (  2, 'second\_test\_repository', TO\_DATE('2024/07/12', 'yyyy/mm/dd')  );  INSERT INTO Repo(id,name,dateCreated) VALUES (  3, 'react\_app', TO\_DATE('2024/07/18', 'yyyy/mm/dd')  );  INSERT INTO Repo(id,name,dateCreated) VALUES (  4, 'sql\_repository', TO\_DATE('2024/07/18', 'yyyy/mm/dd')  );  INSERT INTO Repo(id,name,dateCreated) VALUES (  5, 'cat\_repository', TO\_DATE('2024/07/20', 'yyyy/mm/dd')  ); |
| Issues | INSERT INTO Issues(id,descriptions,dateResolved,repoID) VALUES (  1, 'repo is empty', TO\_DATE('2024/07/08', 'yyyy/mm/dd'), 1  );  INSERT INTO Issues(id,descriptions,dateResolved,repoID) VALUES (  2, 'file A is wrong', TO\_DATE('2024/07/08', 'yyyy/mm/dd'), 1  );  INSERT INTO Issues(id,descriptions,dateResolved,repoID) VALUES (  3, 'bug in GUI', TO\_DATE('2024/07/08', 'yyyy/mm/dd'), 2  );  INSERT INTO Issues(id,descriptions,dateResolved,repoID) VALUES (  4, 'Missing Supabase API key', NULL, 3  );  INSERT INTO Issues(id,descriptions,dateResolved,repoID) VALUES (  5, 'Redirecting wrong in location', NULL, 5  ); |
| Comments | INSERT INTO Issues(id,userid,issueId,message,timePosted) VALUES (  1, 1, 1, 'Will be adding new file soon',  TO\_DATE('2024/07/08 21:02:44', 'yyyy/mm/dd hh24:mi:ss')  );  INSERT INTO Issues(id,userid,issueId,message,timePosted) VALUES (  2, 1, 1, 'Will add two files soon',  TO\_DATE('2024/07/09 07:22:14', 'yyyy/mm/dd hh24:mi:ss')  );  INSERT INTO Issues(id,userid,issueId,message,timePosted) VALUES (  3, 1, 2, 'File A is wrong because of ABC',  TO\_DATE('2024/07/12 10:04:25', 'yyyy/mm/dd hh24:mi:ss')  );  INSERT INTO Issues(id,userid,issueId,message,timePosted) VALUES (  4, 2, 2, 'I’m having trouble understanding why',  TO\_DATE('2024/07/12 14:23:58', 'yyyy/mm/dd hh24:mi:ss')  );  INSERT INTO Issues(id,userid,issueId,message,timePosted) VALUES (  5, 1, 2, 'It’s cause of ABC',  TO\_DATE('2024/07/12 14:28:21', 'yyyy/mm/dd hh24:mi:ss')  ); |
| Commits | INSERT INTO Commits(id,dateCreated,message,repoid,branchName,creatorUserID) VALUES (  1, TO\_DATE('2024/07/09 14:28:21', 'yyyy/mm/dd hh24:mi:ss'), 'added file', 1, 'main', 1  );  INSERT INTO Commits(id,dateCreated,message,repoid,branchName,creatorUserID) VALUES (  2, TO\_DATE('2024/07/09 14:30:00', 'yyyy/mm/dd hh24:mi:ss'), 'edited file', 1, 'main', 1  );  INSERT INTO Commits(id,dateCreated,message,repoid,branchName,creatorUserID) VALUES (  3, TO\_DATE('2024/07/09 14:34:00', 'yyyy/mm/dd hh24:mi:ss'), 'added file 2', 1, 'main', 1  );  INSERT INTO Commits(id,dateCreated,message,repoid,branchName,creatorUserID) VALUES (  4, TO\_DATE('2024/07/19 08:00:00', 'yyyy/mm/dd hh24:mi:ss'), 'init repo', 2, 'main', 2  );  INSERT INTO Commits(id,dateCreated,message,repoid,branchName,creatorUserID) VALUES (  5, TO\_DATE('2024/07/11 08:00:00', 'yyyy/mm/dd hh24:mi:ss'), 'delete file', 1, 'testbranch', 1  ); |
| Branch | INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  1, 'main', NULL, TO\_DATE('2024/07/08', 'yyyy/mm/dd')  );  INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  2, 'main', NULL, TO\_DATE('2024/07/12', 'yyyy/mm/dd')  );  INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  3, 'main', NULL, TO\_DATE('2024/07/18', 'yyyy/mm/dd')  );  INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  4, 'main', NULL, TO\_DATE('2024/07/18', 'yyyy/mm/dd')  );  INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  5, 'main', NULL, TO\_DATE('2024/07/20', 'yyyy/mm/dd')  );  INSERT INTO Branch(repoid,name,fromCommitID,createdOn) VALUES (  1, 'testbranch', 3, TO\_DATE('2024/07/11', 'yyyy/mm/dd')  ); |
| Blob | INSERT INTO Blob (hash, content) VALUES (‘2CF24DBA5FB0A30E26E83B2AC5B9E29E1B161E5C1FA7425E73043362938B9824’,  ‘hello’)  INSERT INTO Blob (hash, content) VALUES (‘EDBB6F552E9C178650D00CF49CC3EE01E2863113441309FBD0301092C2948D0D’,  ‘abcdedghijklm’)  INSERT INTO Blob (hash, content) Values (‘01CE3A013AE8BDFAF5B9F215A728A16FC32B9CE02ACD464D0B6C81EEB968645F’,  ‘Really long file incoming aisdijasdiaijdaidiaodiadiaidasidsaidjaf0wq90iiasdasdasdjiasdoaaiosdaooasdasdgsagagggsdgruwqrqrqweqweiuqwieqwiueiuoqweuiqdasodijoaijosfasifiasfigiuaiuogaisugiaigiaspgiuasipuipasgiaigsaiogiopsagioaigpoapiosgpigasiogaugsagiuagiuaiusdiaisdoaiodiadiaiguw98qeuqu888q9we8qaiifqpwe0qw90e9quw90-eqw90eq-w09ue09q-w982uiqpoidij80-122-081231ufaiospfa080-9qw8eqhfq-we89q8wryhq-89wr8-q8q8-w-814u1hjqosuhiaifhuq8w980r98qw89rhuqwr7uq00a9sdhwq7urqri9’)  INSERT INTO Blob (hash, content) VALUES (‘e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855’,  ‘’)  INSER INTO Blob (hash, content) VALUES (‘DF3B852F0FD6EA481761D2DFD2CBE5479B49F7D48D863CB79D1B54C0C285EC5F‘,‘  #include <iostream>  #include <time.h>  const int n = 3;  const int DSIZE = 10;  const int block\_size = 32;  \_\_global\_\_ void optimizedmmul(float \*a, float \*b, float \*c, int size)  {  int col = threadIdx.x + blockIdx.x \* blockDim.x;  int row = threadIdx.y + blockIdx.y \* blockDim.y;  \_\_shared\_\_ float s\_a[block\_size \* block\_size];  \_\_shared\_\_ float s\_b[block\_size \* block\_size];  int temp = 0;  if ((col < size) && (row < size))  {  s\_a[col + row \* size] = a[col + row \* size];  s\_b[col + row \* size] = b[col + row \* size];  }  \_\_syncthreads();  if ((col < size) && (row < size))  {  for (int i = 0; i < size; i++)  {  temp += s\_a[row \* size + i] \* s\_b[col + i \* size];  }  c[col + row \* size] = temp;  }‘) |
| Files | INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  1, '/', TO\_DATE('2024/07/08', 'yyyy/mm/dd'), ‘‘01CE3A013AE8BDFAF5B9F215A728A16FC32B9CE02ACD464D0B6C81EEB968645F’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  2, '/', TO\_DATE('2024/07/12', 'yyyy/mm/dd'), ‘e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  3, '/', TO\_DATE('2024/07/18', 'yyyy/mm/dd'), ‘e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  4, '/', TO\_DATE('2024/07/18', 'yyyy/mm/dd'), ‘e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855’  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  5, '/', TO\_DATE('2024/07/20', 'yyyy/mm/dd'), ‘e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  6, '/hello.cu', TO\_DATE('2024/07/11', 'yyyy/mm/dd'), ‘DF3B852F0FD6EA481761D2DFD2CBE5479B49F7D48D863CB79D1B54C0C285EC5F’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  7, '/source2/file', TO\_DATE('2024/07/11', 'yyyy/mm/dd'), ‘68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  8, '/source/file', TO\_DATE('2024/07/11', 'yyyy/mm/dd'), '68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728'  );  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  9, '/folder2/emptyFile', TO\_DATE('2024/07/12', 'yyyy/mm/dd'), ‘68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728’  )  INSERT INTO Files(id, path, createdOn, blobHash) VALUES (  10, '/folder2/emptyFile', TO\_DATE('2024/07/13', 'yyyy/mm/dd'), ‘68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728’  ) |
| Folders | INSERT INTO Folders(id, numberOfFiles) VALUE (1, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (2, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (3, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (4, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (5, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (6, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (7, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (9, 0);  INSERT INTO Folders(id, numberOfFiles) VALUE (10, 0); |
| IssuesAssignedTo | INSERT INTO IssuesAssignedTo(userid,issueid) VALUES (1, 1);  INSERT INTO IssuesAssignedTo(userid,issueid) VALUES (1, 2);  INSERT INTO IssuesAssignedTo(userid,issueid) VALUES (1, 3);  INSERT INTO IssuesAssignedTo(userid,issueid) VALUES (4, 4);  INSERT INTO IssuesAssignedTo(userid,issueid) VALUES (5, 4); |
| Permissions | INSERT INTO UserContributesTo(permissions,edit,owner) VALUES (1, false, true);  INSERT INTO UserContributesTo(permissions,edit,owner) VALUES (2, true, true);  INSERT INTO UserContributesTo(permissions,edit,owner) VALUES (3, false, false);  INSERT INTO UserContributesTo(permissions,edit,owner) VALUES (4, true, false);  **Note for Jessica: instructions did say at least 5 tuples but more than 4 would be redundant here** |
| UserContributesTo | INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (1, 1, 2);  INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (1, 2, 2);  INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (4, 3, 3);  INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (5, 3, 4);  INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (3, 4, 2);  INSERT INTO UserContributesTo(userid,repoid, permissions) VALUES (3, 5, 2); |
| CommitsAndFolders | INSERT INTO CommitsAndFolders (folderId, commidId) VALUES (1, 1);  INSERT INTO CommitsAndFolders (folderId, commidId) VALUES (7,1);  INSERT INTO CommitsAndFolders (folderId, commidId) VALUES (2,2);  INSERT INTO CommitsAndFolders (folderId, commidId) VALUES (9,1);  INSERT INTO CommitsAndFolders (folderId, commidId) VALUES (10,1); |
| FilesInFolders | INSERT INTO FilesInFolders(folderId, fileId) VALUES (1,7);  INSERT INTO FilesInFolders(folderId, fileId) VALUES (7,8);  INSERT INTO FilesInFolders(folderId, fileId) VALUES (1,9);  INSERT INTO FilesInFolders(folderId, fileId) VALUES (1,10); |

**Note: Be consistent with the names used in your ER diagram, schema, and FDs. Make a note if the name has been intentionally changed.**