CO527 : Advanced Database Systems

Project Topic : Content management system for conference and research papers

Group Members :

1. *M.M.M Aslam - E/15/021*
2. *M.M.M Irfan - E/15/138*
3. *M.M.M Suhail - E/15/348*

1. Introduction

This project is a Content Management System for managing the research paper submissions for conferences. Moreover, this platform provides a way of buying tickets for conferences. This idea was inspired by a real-world web application called ISABE. The idea of this project is to build a ***SaaS*** platform so that this can be used by multiple clients. But as the initial step, this project focuses on only a single client. Later this can be extended to multiple clients by implementing a ***multi-tenant SaaS*** application. The idea is to have the same database structure for all clients and have different databases for different clients.

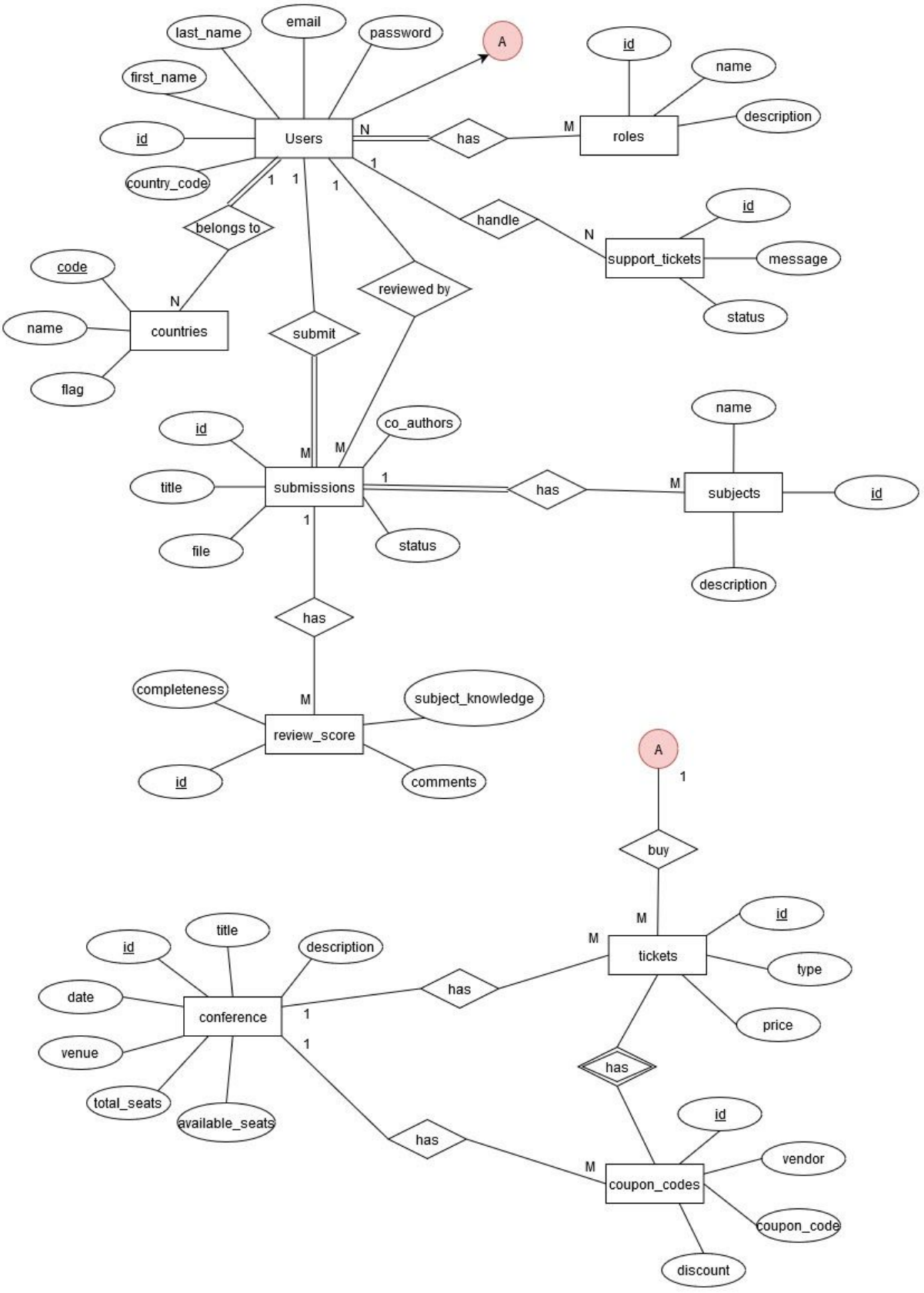
1.1 The functionality of the application

Basically the users will have multiple roles as Admin, reviewer and author, an additional role will be there for Super admins (the developers). Normally a user can sign up as an author first and a super admin can make an author as an admin, then the admin can manage the whole platform there onwards. An author can submit a research paper and the reviewers will evaluate it based on certain criteria. So the submissions can be accepted or rejected according to the evaluation. Moreover, admins can add details of the conference and tickets for the conferences so that the user can buy tickets through our platform.

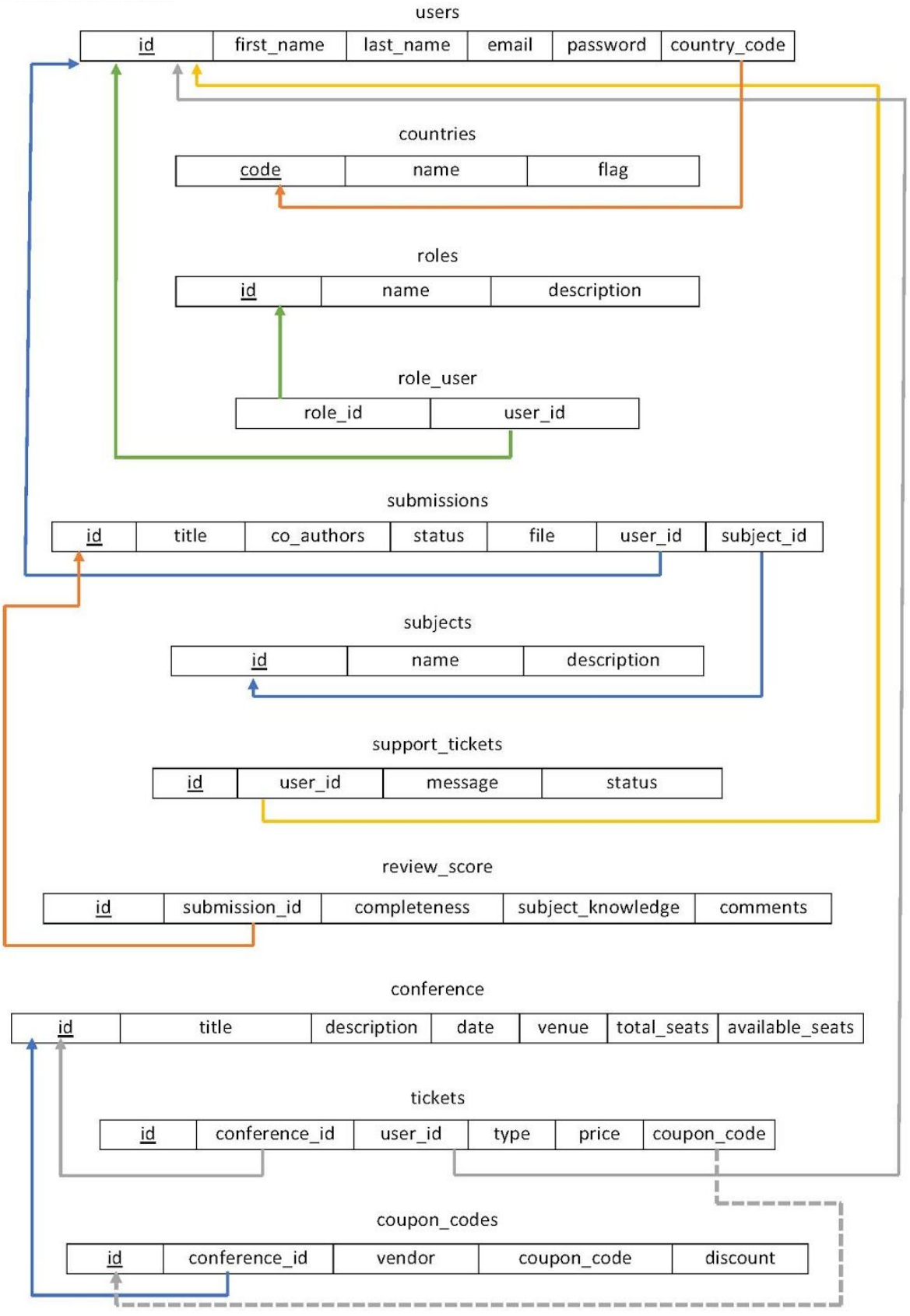
1.2 Scope for a Database project

As it can be seen that this project has a reasonable scope for databases where this will include many relations of the database. And these relations are going to have so many entries where optimization is necessary to get high performance.
Moreover, there are critical sections in the project where using a transaction is a must, for example when reviewing a paper or when buying tickets. These have to be reverted back if something goes wrong in the middle.

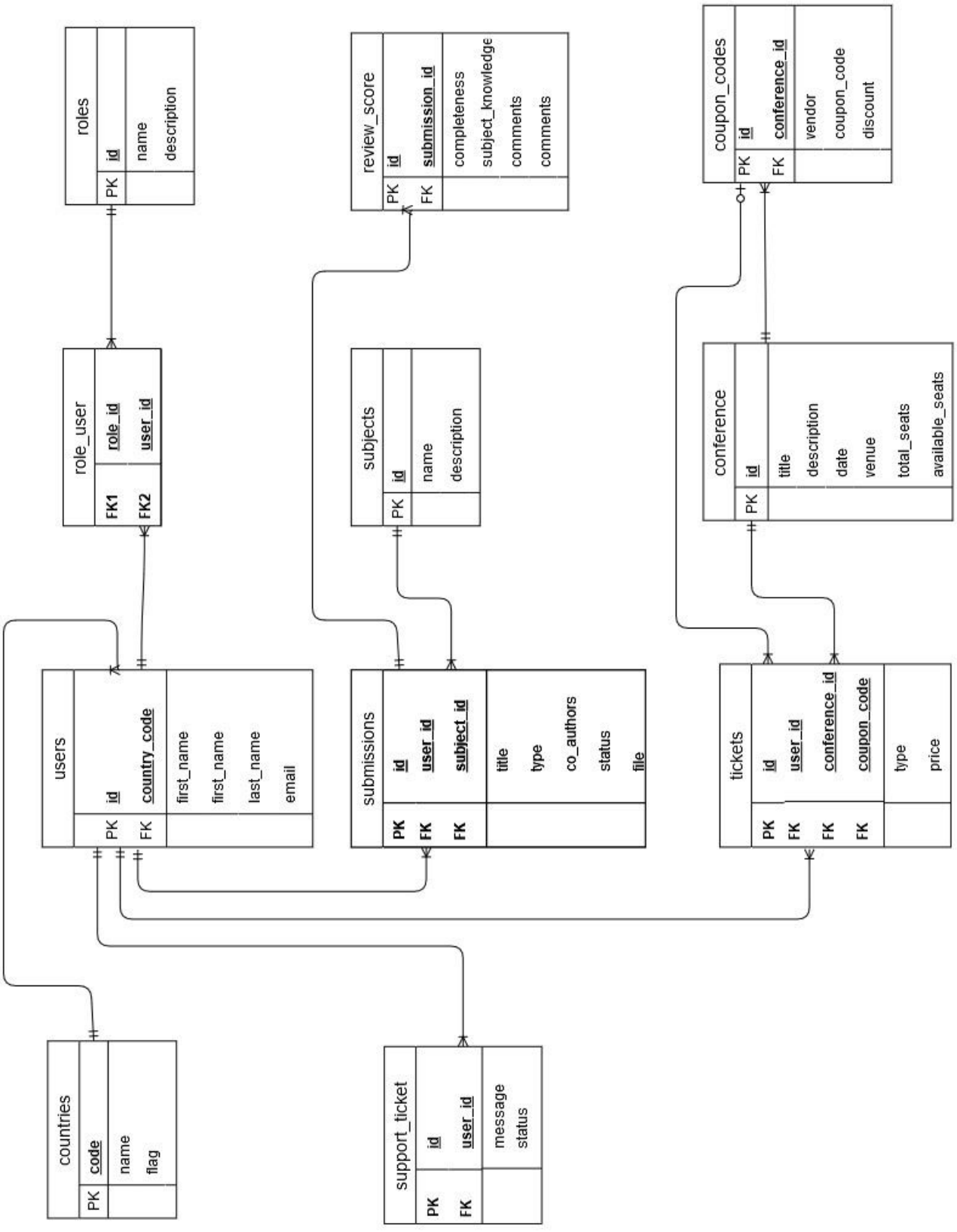
2. Conceptual Design of the database



3. Logical Design of the Database



4. Physical Design of the Database



5. Sample Data

Sample data for each table was created with a ***PHP*** library called faker which is an open-source package used to generate sample data. The database is seeded with the data generated by this library. This is a very useful feature in our development process since this **can generate any amount of data** as we want, this is used to load the database with enough data to test the query performance and use it for query optimization.

Since the project is developed remotely, the Database schema and the test data were separated and used for development. For the Database schema, the SQL script was used to create all the tables required and the test data was handled in a CSV format since there were issues with MySQL versions when we just try to take a dump and import it on another environment. So CSV made the job easier so all environments can load a CSV file.

Following are the number of records in each table with the sample data

- Countries - 242
- Roles - 4
- Subjects - 20
- Users - 10,000
- Role_user (pivot) - 10,000
- Submissions - 54,834
- Review scores - 1000
- Support tickets - 20,040
- Conferences - 100
- Tickets - 5000
- Coupon Codes - 10

Note:

- The csv files can be found in the ***data*** folder
- Code used to generate the seed can be found at https://github.com/irfanm96/CO527-project/blob/master/database/seeds/DatabaseSeeder.php (this is a sample Laravel PHP application, this is only used to generate the seed)

6. Indexes used

While creating indexes, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.
Practically, indexes are also a type of tables, which keep the primary key or index field and a pointer to each record into the actual table.
On the other hand, MySQL indexes **may take up more space and decrease performance on inserts, deletes, and updates**. So they are costly in terms of these parameters. Therefore we created indexes on tables where we will have a lot of entries in the future.

So we selected to use explicit indexes (other than the primary key) on users and submissions tables. Almost all the other tables will not require an index.

- Users table
  Indexed ***first_name*** and ***last_name*** since most of the quires will include this attributes in where, order by or group by queries

- Submissions table
  Indexed the ***title*** of the submission since it will be used as a search column in the queries

Note: Index related SQL queries can be found at ***indexes.sql*** file

7. Query optimization

In the project, Various query optimization techniques are used. Following are some of those

7.1 Use pagination when retrieving data from large tables

Mostly the backend will have a simple CRUD (Create, Read, Update, Delete) operation user interface. Moreover, there will be data tables for most of the tables in the database. So when the number of entries grows up we should have a way of breaking them into pages and load data based on the page.so that we do not need to load the complete dataset from the database which is costly in terms of the database performance as well as the performance of the application.

So we've implemented pagination with the help of LIMIT and OFFSET keywords in MySQL Defining an OFFSET for the query. For example

page 1 - (records 01-10): offset = 0, limit=10;
page 2 - (records 11-20) offset = 10, limit =10;
A sample query from the actual database

```
-- get the users with first name, last name and email page 1
SELECT first_name,last_name,email FROM users LIMIT 10 OFFSET 0;
```

7.2 Index required columns used in 'where', 'order by', and 'group by' Clauses

Apart from guaranteeing uniquely identifiable records, an index allows MySQL server to fetch results faster from a database. An index is also very useful when it comes to sorting records.
Indexes used are discussed in the section above

7.3 Avoid LIKE Expressions With Leading Wildcards

As we know we can use indexes to speed up the query performance, but it comes with a cost. So when optimizing the queries we should choose proper indexes so that we can increase the query execution performance. But we need to make sure that we make sure that the query execution uses those created indexes.

We faced an issue in this manner. We created the indexes on the users table for firstname and lastname field. Since most of the queries will use it in their orderby or where clauses.

When searching the table with the LIKE operator and with wildcard operators in both ends, It was not actually using the indexes that were created.

We used Explain to see the execution plan and it only says that those can be possible keys, but didn't actually use it. The reason was with the wildcard matching (the leading wildcard), MySQL couldn't use the created indexes since we do wildcard matching in both ends so that when a user types a part of the name, we can filter out the results in the UI. So we chose to fix this issue by removing the wildcard at the beginning and made the query with a wildcard at the end of the search term.

With leading wildcard (created indexes were not used)

```
mysql> explain select * from users where first_name like "%john%" or last_name like "%john%";
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | users | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 1000 |    20.99 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

Without leading wildcard (both created indexes are used)

```
mysql> explain select * from users where first_name like "john%" or last_name like "john%";
+----+-------------+-------+------------+-------------+-----------------------------+-----------------------------+---------+------+------+----------+-----------------------------------------------------------+
| id | select_type | table | partitions | type        | possible_keys               | key                         | key_len | ref  | rows | filtered | Extra                                                     |
+----+-------------+-------+------------+-------------+-----------------------------+-----------------------------+---------+------+------+----------+-----------------------------------------------------------+
|  1 | SIMPLE      | users | NULL       | index_merge | idx_firstname,idx_lastname | idx_firstname,idx_lastname | 122,123 | NULL |    9 |   100.00 | Using sort_union(idx_firstname,idx_lastname); Using where |
+----+-------------+-------+------------+-------------+-----------------------------+-----------------------------+---------+------+------+----------+-----------------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

7.4 Making use of MySQL Full-Text Searches

When we need to search data using wildcards and we don't want our database to underperform, we should consider using MySQL full-text search (FTS) because it is far much faster than queries using wildcard characters.
Furthermore, FTS can also bring better and relevant results when we are searching a huge database.So for the users table we used a FTS as below

```
-- this particular query needs to be executed as root or db_admin since
altering involved
-- make use of full text
-- first alter the table
Alter table users ADD FULLTEXT (first_name, last_name);
--- now use match
explain select * from users where match(first_name, last_name) AGAINST
('john');
```

```
mysql> explain select * from users where match(first_name, last_name) AGAINST ('john');
+----+-------------+-------+------------+----------+---------------+------------+---------+-------+------+----------+-----------------------------+
| id | select_type | table | partitions | type     | possible_keys | key        | key_len | ref   | rows | filtered | Extra                       |
+----+-------------+-------+------------+----------+---------------+------------+---------+-------+------+----------+-----------------------------+
|  1 | SIMPLE      | users | NULL       | fulltext | first_name    | first_name | 0       | const |    1 |   100.00 | Using where; Ft_hints: sorted |
+----+-------------+-------+------------+----------+---------------+------------+---------+-------+------+----------+-----------------------------+
1 row in set, 1 warning (0.00 sec)
```

7.5 Select only the required attributes of the relations

As discussed in the class we tried to push the PROJECT statement down in the query tree so that we only retrieve the required data and not retrieve any unwanted columns.
Let's consider an example from the actual database

```
-- select submission title with username and the subject name
select        users.first_name,    users.last_name,    submissions.title,
subjects.name
from submissions,users,subjects
where   submissions.user_id   =   users.id   and   submissions.subject_id   =
subjects.id;
```

Note : More queries can be found in, **queries.sql** file

7.6 Optimized Database Schema

7.6.1 Normalized Tables

In MySQL database normalization, we should represent a fact only once in the entire database.
For example, Don't repeat the user name in every table; instead just use the user_Id for reference in other tables.
So all the tables are normalized in our design according to the above principle, following is an example of it.

```
CREATE TABLE IF NOT EXISTS submissions (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    subject_id INT NOT NULL,
    title VARCHAR(255),
    -- type VARCHAR(20),
    co_authors JSON default NULL,
    status ENUM('pending', 'approved', 'rejected'),
    file VARCHAR(255),
    createdAt timestamp default current_timestamp,
    updatedAt timestamp,
      FOREIGN KEY (user_id) REFERENCES users(id) ON UPDATE CASCADE ON
DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES subjects(id));
```

So it can be seen that both subject and the user are referenced based on it's ID. and the datatype of foreing key should match the data type of the actual table key. That is , if INT is used here for user_id then in users table ID should be in INT type

7.6.2 Avoid Null Values

We should avoid NULL values whenever possible because they can harm the database results. For instance, if we want to get the sum of a column in a database but a particular record has a null value, the expected result might misbehave unless we use MySQL 'ifnull' statement to return an alternative value if a record is null.

So, most of the attributes are not nullable in the database design. But some of them are nullable since they appear on certain times, but as a whole, most of the attributes are not nullable in the design.

Note : queries used to analyse the performance in this topic can be found at ***optimizations.sql*** file

8. Use of Transactions Processing

Transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit. In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

When two sessions try to alter the same data, then the transactions can overlap. So, to overcome that issue, we use transaction processing. Here we are using transaction processing for both selling/buying tickets in the tickets table and rating in the review_scores table.

9. Security

In terms of database administration, security for the tables is achieved by having users with specific privileges. We have defined two users in the database as follows

- db_admin
- backend_dev

Both users have no access to any other database other than the project's DB in the database server, which gives security to all the other databases. Moreover, in the production environment, only encrypted connections are allowed to connect to the database server.

For security reasons and to protect critical data from accidentally being deleted the delete action is purposefully disabled for both the users on **tickets purchased** and the **approved research papers**.

For the tickets table, the objective is achieved simply by not giving DELETE privilege to both the users. But in the submissions table, we need to provide DELETE privilege but we need to restrict the deletion of approved submissions. So this is achieved by throwing an error in a before delete trigger on the submissions table since MYSQL cannot handle privileges to each entry separately

Moreover, it can be seen that in the implementation, the privileges are given for each table in as a separate Grant command. Mysql doesn't support grant on multiple objects.

But we had another option to enable partial revokes so we can first grant all permissions and then revoke the unnecessary parts later. But this is not really recommended by the DB administrators since Once enabled, partial revokes cannot be disabled if any account has privilege restrictions. If any such account exists, disabling partial revokes fails
So we chose to stick to the simple approach since our design has only a few tables to deal with.

Let's consider the privileges of both roles

- db_admin
  db_admin user has almost executive privileges to the project's database, this is the user account which the DB admins use to login to the database and do operations on it. So they can modify almost anything other than the above restrictions. For example, db_admin can create any tables as needed or create triggers or update the database design, create necessary indexes and etc. This is the dedicated role for the database designer and the administrator for the project

- backend_dev
  This user is created so that the developers whoever is developing the backend can use this to interact with the database. With all the restrictions above, some more restrictions are added to this user. For example, this user can retrieve, update or create data from any tables. And other than the delete restriction on the tickets table, this user has the privilege to delete any other entries. This user cannot modify any table structure or change the design since it's all handled by the db_admin, and this user can only do read and write operations as needed. Moreover, this user is being given the permission to create views in the database as needed since they will be useful for the information retrieving

Note : All the implementation of the above security plan can be found in ***security.sql*** file

10. User interface

Currently the application is deployed to a production server and the database is in the AWS RDS services. You can visit the website https://ecsuop2020.web.app/
Login url: https://ecsuop2020.web.app/login
Credentials : Email - escadmin@gmail.com ,Password - 123456@
(The user interface is still under construction)

11. Description on the submission files

➔ **Data folder -** contains the sample data for each table in CSV format
➔ **create_tables.sql -** contains SQL script to create the database
➔ **indexes.sql -** contains SQL script to create indexes
➔ **security.sql -** contains SQL implementation of the security plan
➔ **optimization.sql -** contains SQL scripts to explain optimization techniques used
➔ **final_dump_prod -** contains a current dump of the production database
➔ **queries.sql -** contains queries written for various use cases
➔ **transaction.sql -** contains SQL script for transactions
➔ **backend folder -** contains all the code for backend

All files mentioned above can be found at
https://github.com/irfanm96/CO527-project/tree/master/files

Backend code : https://github.com/aslam021/Server---CMS