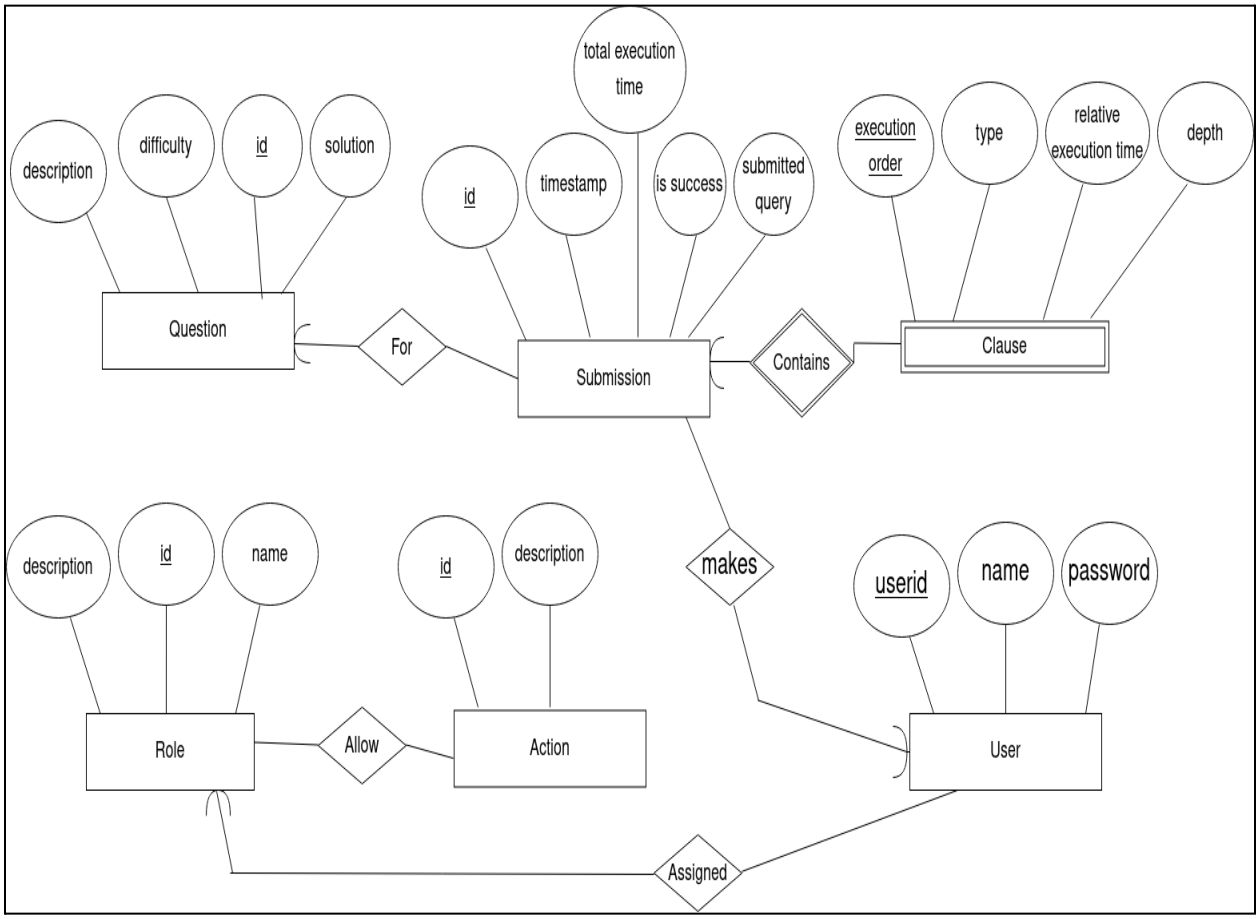
Query Analyzer

ER diagram



Assumptions

1. Each question requires only one DML query to solve. The crux of our application is to produce the intermediate results obtained for a SQL query. Hence in the ER diagram we have a **many-one** relation between **clauses** and **submission** and **many-one** relationship between **submission** and **question**. The clause(eg. FROM, WHERE,ORDER BY) is a weak entity and has an attribute execution order, which is the step number at which the clause is executed with respect to the submission query. The application currently has two roles, one for **Instructor** and other for **Student**. At any point, a user will only be able to use features(actions) available to any one of the roles. We currently have two actions: viewing submission statistics and making submissions. Instructors can do both(make submissions to view how the student journey is). However

in future it could be possible that actions for roles are disjoint. The **many-one** relationship from **user** to **role** and **many-many** relationship from **role** to **action** provides sufficient extensibility. For a new actor, a new role will be created and actions relevant to the role can be referenced using the **many-many** relationship.

Description of relationships/entities and usefulness to application:

There are **six** entities:

1. **Question:** This entity contains details of a question. The table is populated and stored statically and we do not currently plan to provide an interface for instructors to update it, as it isn't the focus of the project. However having this table provides scope for the same in future
    a. id: Auto increment field
    b. solution: Text field containing a correct query.
    c. difficulty: An enum indicating easy, medium and hard.
    d. description: The text of the question along with schema for the tables involved.

    Relationships:
    - A question can have many submissions.

2. **Submission:** A submission is specific to a particular question and contains:
    a. id: Auto increment field
    b. total execution time: Numeric field indicating the amount of time the query took to execute on a SQL compiler.
    c. is success: binary field indicating if the submission was successful(to measure success we will execute the query linked to the submission's question against the database and compare the results against the student's query).
    d. timestamp: Date time field which indicates submission date and time.
    e. submitted query: Text field containing the pretty printed version of the query submitted by the student. Enables instructors to pull out actual queries if they find the metadata to be interesting.

    Relationships:
    - A submission is linked/is for a particular question.
    - A user can make multiple submissions. Each submission tracks the question it is for. This does not restrict the user in any way. He/she can make one or more submissions for each question.
3. **Clause:** It is a clause belonging to one student submission query on a question. The clause can be one of FROM, WHERE, SELECT, GROUP BY, HAVING, ORDER BY. It is a weak entity as it does not have unique meaning without the submission with which it is associated. Clause contains:
    a. execution order: Given a query, it is a number denoting when a particular clause within it gets executed. Numbering starts from 1 for clauses of a submission.
    b. type: Text field denoting clause type like FROM, WHERE, GROUP BY…

c. relative execution time: It is the ratio of execution time for the clause to the sum of execution times for all clauses belonging to the query. It is possible that the SQL compiler internally creates a plan looking at all clauses and optimizes across them making the total execution time lesser than sum of times of the individual clauses. However, the intention of the fraction is to not give an accurate number but an idea to students as to which step will be the bottleneck.
d. depth: It is set to one, if the clause is part of the outermost query. It increments by 1 as we move into clauses within a subquery. This can be used to analyze if there are interesting aspects related to nested and unnested clauses.

Relationships:

- A submission is the supporting entity for clause and “contains” is the supporting relationship. Each clause is part of 1 submission. A submission query can have many clauses.

4. **User:**
    a. userId: A statically stored string as we are not planning for signup functionality currently.
    b. password: A hash of the password will be stored for security purposes.
    c. name: User’s name which is used for reporting purposes as well as injection into UI.
5. **Role:**
    a. id: Auto increment field.
    b. name: Text field specifying role name such as instructor/student.
    c. description: Text field indicating what the role means with respect to the. application. This will be pulled into the UI allowing for updates without a code change.

Relationships:

- Each user is assigned one role, but a role can be assigned to many users. For example, there can be many students.

6. **Action:**
    a. id: Auto increment field.
    b. description: Text field to describe what the action will allow the user to do with respect to the application. For example, “the view dashboard action will allow the instructors to see submission statistics aggregated per question or per student. It will also allow sorting of columns.” This will be pulled into the UI allowing for updates without a code change.

Relationships:

- An action can be assigned to many roles and a role can be allowed to perform many actions. Instructors can view dashboards as well as submit solutions. Both students and instructors can submit solutions.

Relationships and cardinalities:

Submission to question via “for” relationship: many to 1
Submission to clause via “contains” relationship: 1 to many
Submission to user via “makes” relationship: many to 1
User to role via “assigned” relationship: many to 1
Role to action via “allow” relationship: many to many

Description of relationships is presented here

Relational Schema:

1. **Question**(id: INT PK,
solution: VARCHAR(300),
difficulty: ENUM(‘easy’,’medium’,’hard’),
description: VARCHAR(500))

2. **Submission**(id: INT PK,
timestamp: DATETIME,
totalExecutionTime: REAL,
isSuccess: BOOLEAN,
submittedQuery: VARCHAR(500),
questionID: INT FK to Question.id
userId: VARCHAR(15) FK toUser.userid
)

3. **Role**(id: INT PK,
name: VARCHAR(30),
description: VARCHAR(200))

4. **Action**(id: INT PK,
description: VARCHAR(200));

5. **User**(userid: VARCHAR(15) PK,
name: VARCHAR(40),
password: VARCHAR(100)
roleid: INT FK to Role.id)

Kept password as 100 characters max length as hash may be long

6. **Allow**(Roleid: INT FK to Role.id PK,
Actionid: INT FK to Action.id PK) -> PK is (Roleid,Actionid)

7. **Clause**(executionOrder: INT PK,
submissionId: INT FK to Submission.id PK,
type: VARCHAR(10),
relativeExecutionTime: REAL,
Depth: INT); -> PK is (executionOrder,submission ID)