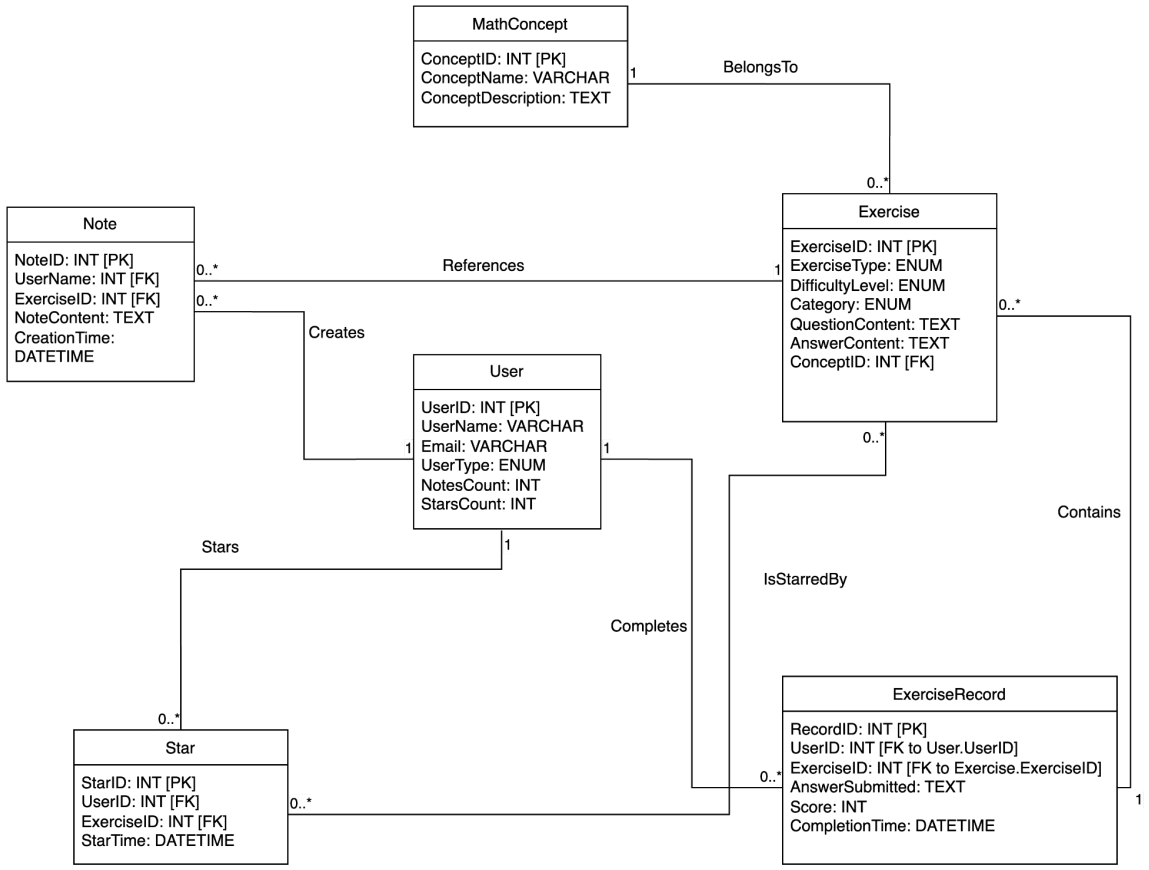
Here is the UML diagram of our database design.



1. Below are the assumptions and explanations for each entity in the mathematics exercise management platform:

User

Assumption: The User entity represents all platform users, including students, teachers, and administrators. Since user information is critical and unique, it is modeled as a separate entity instead of being attributes of other entities.

The User entity stores essential user details like username, email, and user type, as well as the count of notes and stars associated with each user.

MathConcept

Assumption: Math concepts categorize exercises into various domains (e.g., Geometry, Algebra). Each math concept has a distinct name and description, making it necessary to create a separate entity.

This entity enables a many-to-one relationship between exercises and math concepts, facilitating exercise categorization and management.

Exercise

Assumption: Exercises are the core entity of the platform, representing all math problems. Each exercise has attributes such as type, difficulty level, and category, and it is associated with other entities like Note, Star, and ExploreHistory.

Because exercises are linked to multiple entities and have distinct attributes, they are modeled as an independent entity.

Note
Assumption: Notes record a user's thoughts and comments on an exercise. Since notes have unique content and creation time, they are modeled as a separate entity rather than an attribute of the Exercise.
This entity records a 1:N relationship between users and exercises, showing which users have created notes on which exercises.

2.Below is the description of relationships.

User to Note:
Relationship Name: Creates
Cardinality: 1:N (One user can create multiple notes, and each note is created by one user).

User to Star:
Relationship Name: Stars
Cardinality: 1:N (One user can star multiple exercises, but each star entry is associated with one user).

User to ExerciseRecord:
Relationship Name: Completes
Cardinality: 1:N (One user can complete multiple exercises, but each exercise record is linked to one user).

Exercise to MathConcept:
Relationship Name: Belongs to
Cardinality: N:1 (Multiple exercises can belong to one math concept, but each exercise only belongs to one concept).

User to ExploreHistory:
Relationship Name: Browses
Cardinality: 1:N (One user can have multiple browsing history entries, but each entry is linked to one user).

Exercise to Star:
Relationship Name: Is Starred by
Cardinality: M:N (Many users can star many exercises, forming a many-to-many relationship).

3.Nomalization

We applied **BCNF (Boyce-Codd Normal Form)** normalization to our database structure to minimize redundancy and eliminate dependencies that could lead to anomalies. BCNF ensures that all non-key attributes are fully functionally dependent on a candidate key, thus preventing partial and transitive dependencies.

Conceptual Analysis: In our database design, BCNF was used to maintain data integrity and prevent common update, insert, and delete anomalies. For example, in our initial design, User and ExerciseRecord were combined in a single table, which caused issues because

UserID alone could not uniquely identify the other attributes in ExerciseRecord (such as Score or AnswerSubmitted). This led to data redundancy and potential update anomalies. For instance, if a user completed multiple exercises and the user's email address changed, we would have to update the email information in multiple records. This would not only be inefficient but could also result in inconsistent data.

Design Adjustments: To resolve these issues, we separated ExerciseRecord into its own table and introduced UserID and ExerciseID as foreign keys, forming a composite key. This ensures that all attributes in ExerciseRecord depend on the combined primary key (UserID + ExerciseID), eliminating partial dependencies.

A similar adjustment was made for Star and Note tables. These relationships were split into separate tables with UserID and ExerciseID as foreign keys to uniquely identify each record. This approach eliminated redundancy and prevented potential anomalies during updates.

After applying BCNF normalization, our database schema is free from redundancy and maintains data consistency. Each attribute is fully functionally dependent on its candidate key, ensuring that there are no partial or transitive dependencies. This design significantly simplifies data management and ensures that the database adheres to best practices.

5.Logical design (relational schema)

User(
UserID: INT [PK],
Username: VARCHAR(50),
Email: VARCHAR(100),
UserType: ENUM('Student', 'Teacher', 'Admin'),
NotesCount: INT,
StarsCount: INT
)

MathConcept(
ConceptID: INT [PK],
ConceptName: VARCHAR(100),
ConceptDescription: TEXT
)

Exercise(
ExerciseID: INT [PK],
ExerciseType: ENUM('Multiple Choice', 'Fill in the Blank', 'Short Answer'),
DifficultyLevel: ENUM('Very Easy', 'Easy', 'Medium', 'Hard', 'Very Hard'), Category:
ENUM('Geometry', 'Algebra', 'Calculus', 'Statistics'),
QuestionContent: TEXT,
AnswerContent: TEXT,
ConceptID: INT [FK to MathConcept.ConceptID]
)

Note(

```
    NoteID: INT [PK],
    UserID: INT [FK to User.UserID],
    ExerciseID: INT [FK to Exercise.ExerciseID],
    NoteContent: TEXT,
    CreationTime: DATETIME
)

Star(
    StarID: INT [PK],
    UserID: INT [FK to User.UserID],
    ExerciseID: INT [FK to Exercise.ExerciseID],
    StarTime: DATETIME
)

ExerciseRecord(
    RecordID: INT [PK],
    UserID: INT [FK to User.UserID],
    ExerciseID: INT [FK to Exercise.ExerciseID],
    AnswerSubmitted: TEXT,
    Score: INT,
    CompletionTime: DATETIME
)

ExploreHistory(
    HistoryID: INT [PK],
    UserID: INT [FK to User.UserID],
    ExerciseID: INT [FK to Exercise.ExerciseID],
    ExploreTime: DATETIME,
    SearchContent: TEXT
)
```