DB Design Report

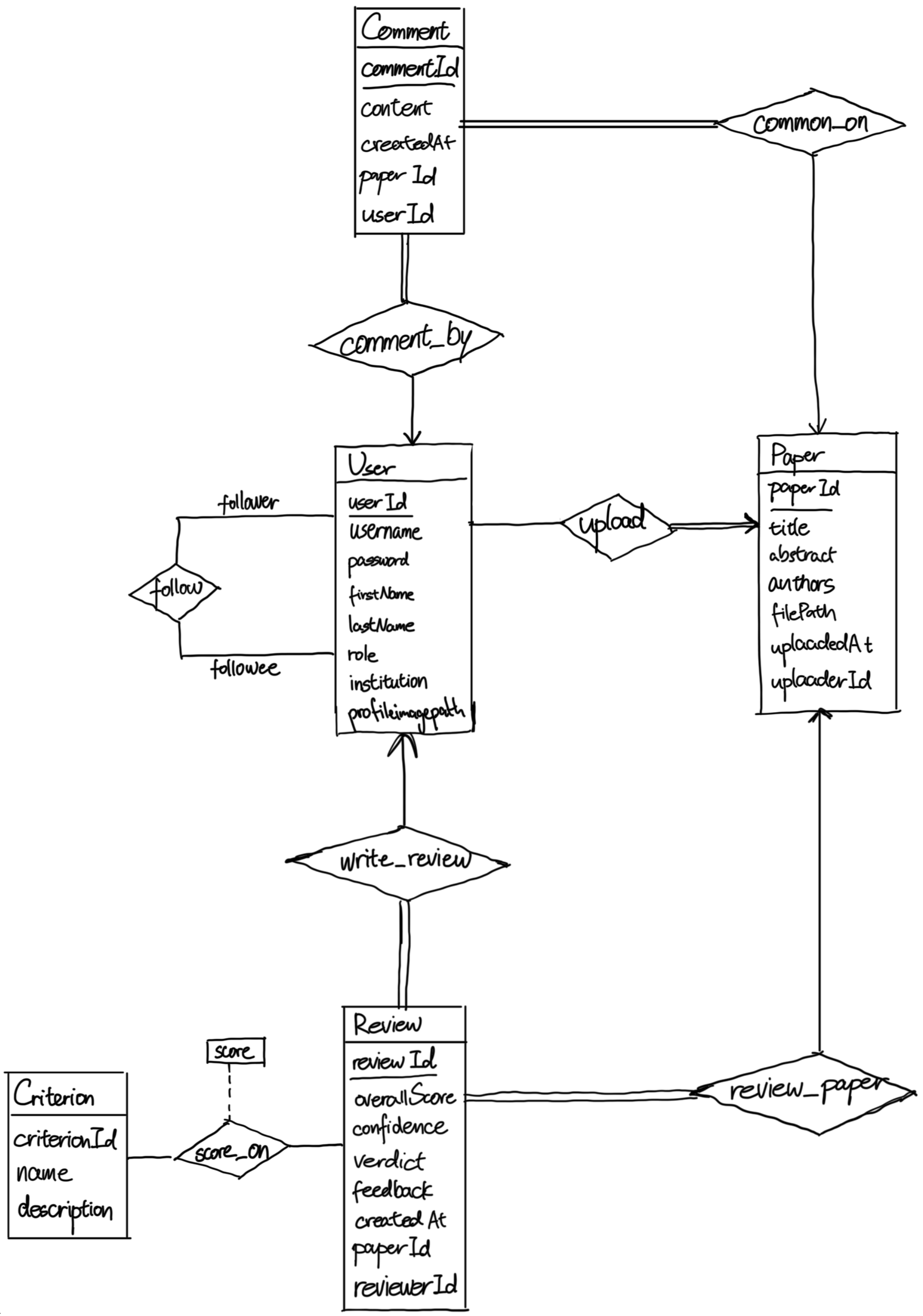
1 Application Overview

Our application is an Open Research Review Platform that supports open-style peer review for research papers, with a strong focus on authors and social interactions.

Main Features

- **Authors**
  - Browse a page showing all authors in the database.
  - Search authors by name.
  - Click an author to view their profile (bio + papers).
  - Follow / unfollow authors.
- **Homepage & User Roles**
  - Homepage prioritizes papers from followed authors.
  - Followed authors and their papers are visually highlighted.
  - Reviewers do not see "Upload Paper".
  - The role attribute controls visible functions.
- **Papers, Reviews, Discussions**
  - Users upload papers (PDFs) with metadata.
  - Reviewers write formal reviews (score, confidence, verdict).
  - Papers include a **Discussion** tab implemented through comments.

The database is designed for scalability and efficient queries supporting author-centered feeds, paper review workflows, and discussions.



sshi

Figure 1: ER-Model

2 ER Model

2.1 Entity Sets and Attributes

Primary keys are underlined.

1. **User**
   - userId (PK), username (unique), password, firstName, lastName, role $\in \{author, reviewer, admin\}$, institution, profileImagePath
2. **Paper**
   - paperId, title, abstract, authors (text), filePath, uploadedAt, uploaderId (FK $\rightarrow$ User)
3. **Review**
   - reviewId, overallScore, confidence, verdict, feedback, createdAt, paperId (FK), reviewerId (FK)
4. **Comment**
   - commentId, content, createdAt, paperId (FK), userId (FK)
5. **Criterion**
   - criterionId, name (unique), description
6. **ReviewCriterionScore** (associative entity)
   - id, reviewId (FK), criterionId (FK), score
7. **Follow** (recursive relationship)
   - followerId, followeeId (FKs to User)

2.2 Relationship Sets and Cardinalities

2.2.1 Upload (User–Paper, many-to-one)

- One User uploads many Papers.
- Each Paper is uploaded by exactly one User.
- ER notation: single line from User; double line + arrow toward Paper.

2.2.2 WriteReview (User–Review, one-to-many)

- One User writes many Reviews.
- Each Review has one User (total participation).

2.2.3 ReviewPaper (Paper–Review, one-to-many)

- One Paper can have many Reviews.
- Each Review evaluates one Paper (total participation).

2.2.4 Follow (User–User, many-to-many recursive)

- A User may follow many users.
- A User may be followed by many users.
- ER notation: one User rectangle, one "follow" diamond, two role-labeled lines: follower / followee.

2.2.5 Comments (User–Comment–Paper)

Two separate relationships:

- comment_by (Comment–User): one user writes many comments.
- comment_on (Comment–Paper): one paper has many comments.

2.2.6 ScoreByCriterion (Review–Criterion, many-to-many)

- Relationship attribute: score.
- Implemented as ReviewCriterionScore.

3 How to Draw the ER Diagram

Follow the standard notation shown in course slides:

1. **Layout**
   - User (top-left), Paper (top-right), Review (center), Comment (bottom-right), Criterion (bottom-left).
2. **User–Paper–Review**
   - upload diamond between User–Paper
   - write_review between User–Review
   - review_paper between Paper–Review
3. **Recursive Follow**
   - One follow diamond

- Two lines labeled “follower” and “followee”

4. **Comments**
   - comment_by between Comment–User
   - comment_on between Comment–Paper
5. **Scoring**
   - score_on diamond between Review–Criterion
   - score attribute attached to the diamond

4 Relational Schema (MySQL)

4.1 USER

```
USER(
  userId INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  firstName VARCHAR(255) NOT NULL,
  lastName VARCHAR(255) NOT NULL,
  role ENUM('author','reviewer','admin') DEFAULT 'author',
  institution VARCHAR(255),
  profileImagePath VARCHAR(255) DEFAULT '/avatars/default.png'
);
```

4.2 FOLLOW

```
FOLLOW(
  followerId INT NOT NULL,
  followeeId INT NOT NULL,
  PRIMARY KEY (followerId, followeeId),
  FOREIGN KEY (followerId) REFERENCES USER(userId)
    ON DELETE CASCADE,
  FOREIGN KEY (followeeId) REFERENCES USER(userId)
    ON DELETE CASCADE
);
```

4.3 PAPER

```
PAPER(
  paperId INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(255) NOT NULL,
  abstract TEXT NOT NULL,
  authors VARCHAR(500) NOT NULL,
  uploaderId INT NOT NULL,
  filePath VARCHAR(500) NOT NULL,
  uploadedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (uploaderId) REFERENCES USER(userId)
);
```

4.4 REVIEW

```
REVIEW(
  reviewId INT PRIMARY KEY AUTO_INCREMENT,
  paperId INT NOT NULL,
  reviewerId INT NOT NULL,
  overallScore INT CHECK (overallScore BETWEEN 1 AND 10),
  confidence INT CHECK (confidence BETWEEN 1 AND 5),
  verdict ENUM('accept','weak_accept','borderline',
               'weak_reject','reject') NOT NULL,
  feedback TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (paperId) REFERENCES PAPER(paperId),
  FOREIGN KEY (reviewerId) REFERENCES USER(userId)
);
```

4.5 COMMENT

```
COMMENT(
  commentId INT PRIMARY KEY AUTO_INCREMENT,
  paperId INT NOT NULL,
  userId INT NOT NULL,
  content TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (paperId) REFERENCES PAPER(paperId),
  FOREIGN KEY (userId) REFERENCES USER(userId)
);
```

4.6 CRITERION

```
CRITERION(
  criterionId INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) UNIQUE NOT NULL,
  description TEXT
);
```

4.7 REVIEW_CRITERION_SCORE

```
REVIEW_CRITERION_SCORE(
  id INT PRIMARY KEY AUTO_INCREMENT,
  reviewId INT NOT NULL,
  criterionId INT NOT NULL,
  score INT CHECK (score BETWEEN 1 AND 5),
  FOREIGN KEY (reviewId) REFERENCES REVIEW(reviewId)
    ON DELETE CASCADE,
  FOREIGN KEY (criterionId) REFERENCES CRITERION(criterionId)
);
```

4.8 Wide Table Construction

To apply normalization theory, we begin by constructing a single wide relation R that contains all attributes used in the application. This hypothetical table represents the result of joining Users, Papers, Reviews, Comments, Criteria, and Criterion Scores into a single denormalized structure:

```
R(
  userId, username, password, firstName, lastName, role, institution, profileImagePath,
  paperId, title, abstract, authors, uploaderId, filePath, uploadedAt,
  reviewId, reviewerId, overallScore, confidence, verdict, feedback, reviewCreatedAt,
  criterionId, criterionName, criterionDescription, score,
  commentId, commentUserId, commentContent, commentCreatedAt
)
```

This design is highly redundant:

- User attributes repeat for every paper, review, comment, and score associated with that user.
- Paper attributes repeat for every review and comment of that paper.
- Criterion attributes repeat for every review-criterion pair.
- Review attributes repeat for every criterion score.
- Comments are repeated whenever joined with reviews or criterion scores.

Because of the partial and transitive dependencies present in this structure, the wide table violates 2NF, 3NF, and BCNF, motivating decomposition into well-structured relations based on functional dependencies.

5 Functional Dependencies

5.1 USER

- userId → all other attributes
- username → all other attributes

5.2 FOLLOW

- (followerId, followeeId) is the only key and determines the tuple.

5.3 PAPER

- paperId → title, abstract, authors, uploaderId, filePath, uploadedAt

5.4 REVIEW

- reviewId → paperId, reviewerId, scores, verdict, etc.

5.5 COMMENT

- commentId → paperId, userId, content, createdAt

5.6 CRITERION

- criterionId → name, description
- name → criterionId, description

5.7 REVIEW_CRITERION_SCORE

- id → reviewId, criterionId, score
- (reviewId, criterionId) → score

6 Normalization Steps and Final Normalized Relations

6.1 Normalization Steps (From Wide Table to BCNF)

Based on the functional dependencies identified in Section 5, we decompose the wide relation **R** to eliminate redundancy and anomalies.

Step 1: Remove user-related partial dependencies

Since

$$\texttt{userId} \rightarrow \texttt{username, password, firstName, lastName, role, institution, profileImagePath},$$

all user attributes depend solely on `userId`, not on the full key of the wide table.

Thus, we extract:

`User(userId, username, password, firstName, lastName, role, institution, profileImagePath)`

Step 2: Remove paper-related partial dependencies

Because

$$\texttt{paperId} \rightarrow \texttt{title, abstract, authors, uploaderId, filePath, uploadedAt},$$

we extract:

`Paper(paperId, title, abstract, authors, uploaderId, filePath, uploadedAt)`

Step 3: Remove review-related dependencies

Since

$$\texttt{reviewId} \rightarrow \texttt{paperId, reviewerId, overallScore, confidence, verdict, feedback, createdAt},$$

we extract:

`Review(reviewId, paperId, reviewerId, overallScore, confidence, verdict, feedback, createdAt)`

Step 4: Remove criterion dependencies

Because

$$\texttt{criterionId} \rightarrow \texttt{criterionName, criterionDescription},$$

we extract:

`Criterion(criterionId, name, description)`

Step 5: Resolve many-to-many Review–Criterion scores

The dependency

$$(\texttt{reviewId}, \texttt{criterionId}) \rightarrow \texttt{score}$$

forms the composite-key entity:

`ReviewCriterionScore(id, reviewId, criterionId, score)`

Step 6: Extract comments

From

$$\texttt{commentId} \rightarrow \texttt{paperId, userId, content, createdAt},$$

we extract:

`Comment(commentId, paperId, userId, content, createdAt)`

Conclusion

After decomposition, all resulting relations satisfy **BCNF** because every functional dependency has a candidate key on its left-hand side. This structure eliminates redundancy and update anomalies while preserving all information.

7 Comparison of ER-Based Schema and Normalized Schema

Both the ER-based design (Section 2) and the normalization process (Sections 5–6) produce the same set of relations:

- `User`
- `Paper`
- `Review`
- `Comment`
- `Follow`
- `Criterion`
- `ReviewCriterionScore`

The tables obtained through normalization validate that the ER model already separates independent entities according to their keys and avoids partial or transitive dependencies. The ER model therefore implicitly satisfies **3NF**, and most relations also satisfy **BCNF**.

Because both approaches produce equivalent schemas, we adopt the **ER-derived schema** as the final database design. This schema is easier to understand, directly reflects the application semantics, and is already fully normalized without unnecessary decomposition.

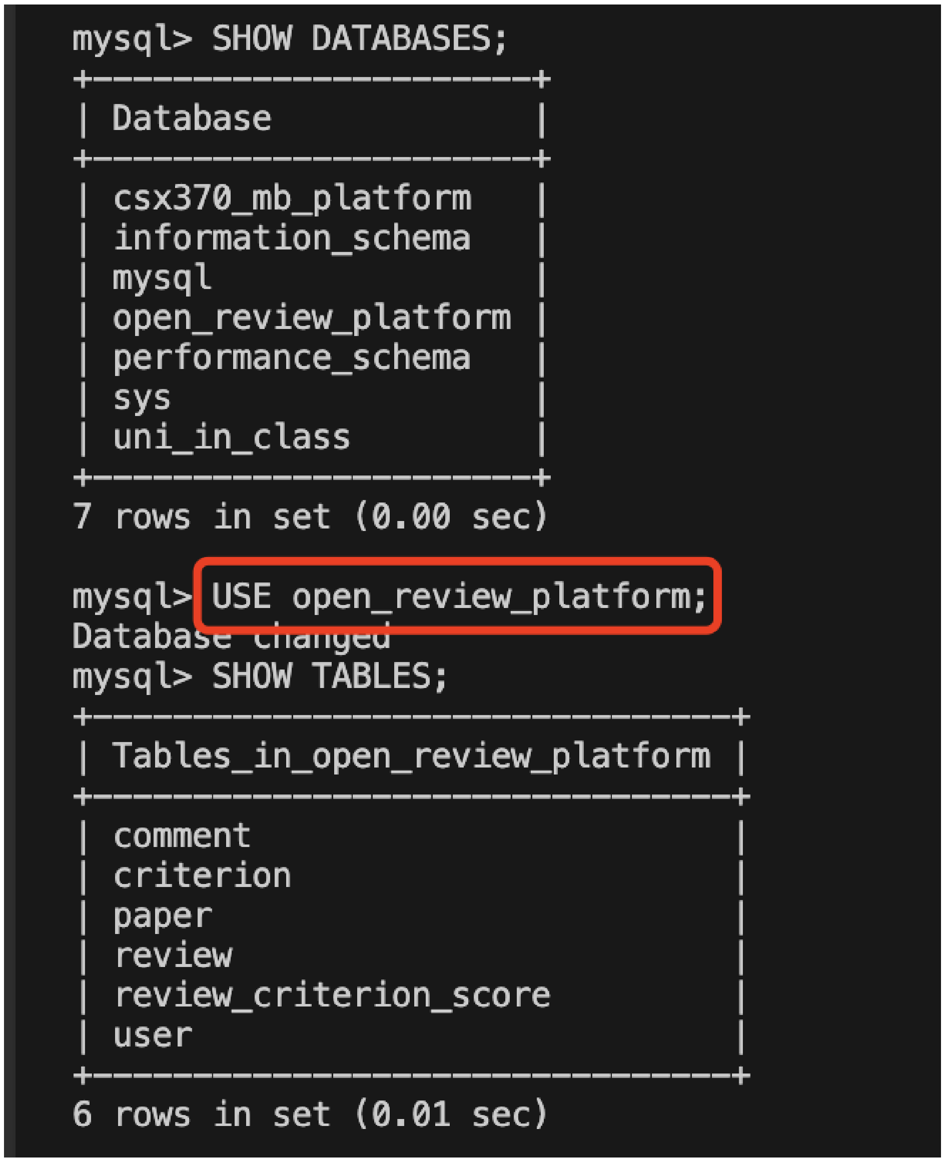
8 MySQL Implementation in Docker

The final schema was implemented inside the MySQL server running in the course-provided Docker container.

8.1 Creating the Schema

The DDL statements from Section 4 were executed in the Docker container:

```
docker exec -it mysql-container mysql -u root -p
USE open_review_platform;
SOURCE ddl.sql;
```



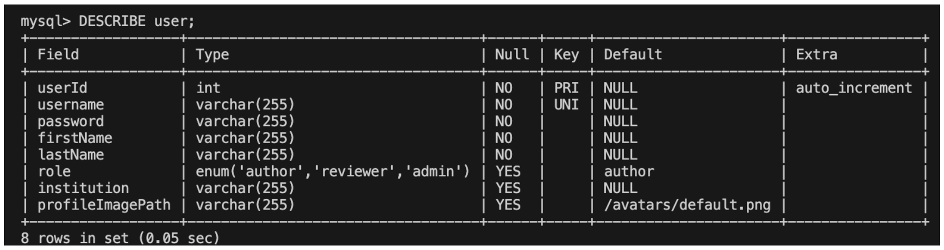
```
mysql> SHOW DATABASES;
+-----------------------+
| Database              |
+-----------------------+
| csx370_mb_platform    |
| information_schema    |
| mysql                 |
| open_review_platform  |
| performance_schema    |
| sys                   |
| uni_in_class          |
+-----------------------+
7 rows in set (0.00 sec)

mysql> USE open_review_platform;
Database changed
mysql> SHOW TABLES;
+--------------------------------+
| Tables_in_open_review_platform |
+--------------------------------+
| comment                        |
| criterion                      |
| paper                          |
| review                         |
| review_criterion_score         |
| user                           |
+--------------------------------+
6 rows in set (0.01 sec)
```

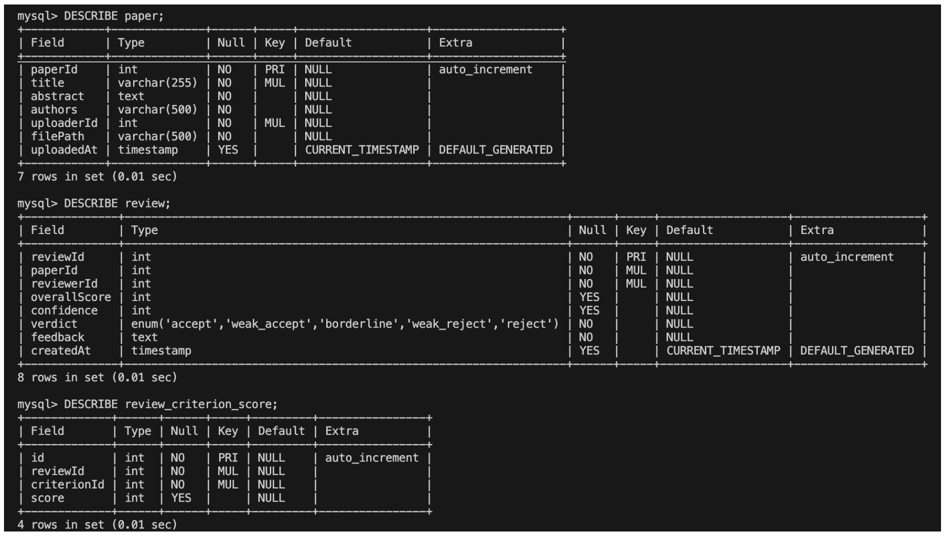
8.2 Verification

To verify the schema, we executed:

```
SHOW TABLES;
DESCRIBE user;
DESCRIBE paper;
DESCRIBE review;
```



```
mysql> DESCRIBE user;
+------------------+-----------------------------------+------+-----+---------------------+----------------+
| Field            | Type                              | Null | Key | Default             | Extra          |
+------------------+-----------------------------------+------+-----+---------------------+----------------+
| userId           | int                               | NO   | PRI | NULL                | auto_increment |
| username         | varchar(255)                      | NO   | UNI | NULL                |                |
| password         | varchar(255)                      | NO   |     | NULL                |                |
| firstName        | varchar(255)                      | NO   |     | NULL                |                |
| lastName         | varchar(255)                      | NO   |     | NULL                |                |
| role             | enum('author','reviewer','admin') | YES  |     | author              |                |
| institution      | varchar(255)                      | YES  |     | NULL                |                |
| profileImagePath | varchar(255)                      | YES  |     | /avatars/default.png |               |
+------------------+-----------------------------------+------+-----+---------------------+----------------+
8 rows in set (0.05 sec)
```



```
mysql> DESCRIBE paper;
+------------+--------------+------+-----+-------------------+-------------------+
| Field      | Type         | Null | Key | Default           | Extra             |
+------------+--------------+------+-----+-------------------+-------------------+
| paperId    | int          | NO   | PRI | NULL              | auto_increment    |
| title      | varchar(255) | NO   | MUL | NULL              |                   |
| abstract   | text         | NO   |     | NULL              |                   |
| authors    | varchar(500) | NO   |     | NULL              |                   |
| uploaderId | int          | NO   | MUL | NULL              |                   |
| filePath   | varchar(500) | NO   |     | NULL              |                   |
| uploadedAt | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+--------------+------+-----+-------------------+-------------------+
7 rows in set (0.01 sec)

mysql> DESCRIBE review;
+--------------+-------------------------------------------------------------------+------+-----+-------------------+-------------------+
| Field        | Type                                                              | Null | Key | Default           | Extra             |
+--------------+-------------------------------------------------------------------+------+-----+-------------------+-------------------+
| reviewId     | int                                                               | NO   | PRI | NULL              | auto_increment    |
| paperId      | int                                                               | NO   | MUL | NULL              |                   |
| reviewerId   | int                                                               | NO   | MUL | NULL              |                   |
| overallScore | int                                                               | YES  |     | NULL              |                   |
| confidence   | int                                                               | YES  |     | NULL              |                   |
| verdict      | enum('accept','weak_accept','borderline','weak_reject','reject') | NO   |     | NULL              |                   |
| feedback     | text                                                              | NO   |     | NULL              |                   |
| createdAt    | timestamp                                                         | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+--------------+-------------------------------------------------------------------+------+-----+-------------------+-------------------+
8 rows in set (0.01 sec)

mysql> DESCRIBE review_criterion_score;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| id          | int  | NO   | PRI | NULL    | auto_increment |
| reviewId    | int  | NO   | MUL | NULL    |                |
| criterionId | int  | NO   | MUL | NULL    |                |
| score       | int  | YES  |     | NULL    |                |
+-------------+------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```