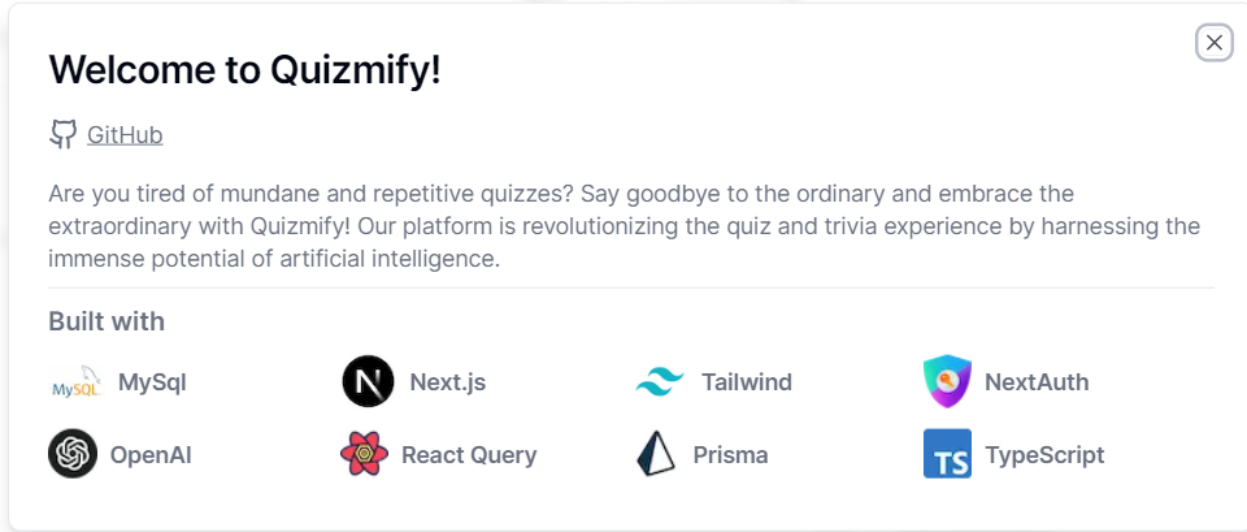
DBMS Project



TEAM MEMBERS

Dheeraj Raj Appikonda - 2110110119 - ar333

Haradeep Pidaka - 2110110229 - hp976

Mani Shankar - 2110110341 - nm147

Objectives and Features:

Quizmify is an AI-powered quiz generator that aims to:

- Enhance Learning and Understanding: Provide users with engaging and personalized quizzes to enhance their learning and understanding of various topics.
- Insights and Solutions: Provide users with insights into their performance, highlighting areas of strength and weakness, along with detailed explanations and solutions for incorrect answers.
- Accessibility and Adaptability: Design the app to be accessible to users of all ages and skill levels, adapting to their individual learning needs.
- Personalized Quiz Generation: Generate quizzes tailored to the user's specific needs and preferences, including the desired topic, number of questions, and difficulty level.
- Gamified Learning Experience: Incorporate gamification elements to make the learning process more engaging and enjoyable, such as points, badges, and leaderboards.

Procedure:

- Topic Selection: Users select the topic they want to be quizzed on from a wide range of available subjects.
- Quiz Customization: Users specify the desired number of questions and difficulty level to tailor the quiz to their needs.
- Quiz Generation: The AI algorithm generates a personalized quiz based on the selected topic, number of questions, and difficulty level.
- Quiz Delivery: Users take the generated quiz, answer multiple-choice questions, and receive immediate feedback.
- Performance Analysis: Upon completion, users receive a detailed analysis of their performance, highlighting areas of strength and weakness.
- Insights and Explanations: For incorrect answers, the app provides detailed explanations and solutions to help users understand the correct concepts.
- Gamification Elements: Points, badges, and leaderboards motivate users to continue learning and improve their performance.
- Continuous Improvement: The AI algorithm continuously learns from user interactions and feedback, refining its ability to generate personalized and effective quizzes.

Entities and Relationships:

Entities:

- User:

Represents individuals who use the system to participate in the quizzes.

Includes Attributes like user ID, email, Name, and Profile Image.

- Game:

Represents a quiz where it contains the topic of the quiz, when the quiz is started and ended, type of the quiz.

Includes attributes like Game ID, user ID, Topic, Game Type, Time Started, and Time Ended.

- Questions:

Represents the questions on a topic chosen by the user.

Includes Attributes like Question ID, Question, QuestionType, Answer, User Answer, option, IsCorrect, PercentageCorrect.

- TopicCount:

Represents all the topics selected by the user.

Includes Attributes like Topic ID, Topic, and Count.

- Session:

Represents the time whenever the user logs into the browser.

Contains Session ID, Session Token, Session Expiry, User ID.

Relationship:

- User_creates_acc:

Represents the users who create an account to log in.

- user_logsin_session

Represents the users who logs in to a session

- User_starts_game

Represents the user who starts a game

- User_answers_questions

Represents the user can answer questions

- Game_contains_questions

Represents the game contains questions

- game_has_topicCount

Represents the game has topicCount

Cardinalities:

User to Account: One-to-One (One User can create only one account)

User to Session: One-to-many (one user can log into many sessions)

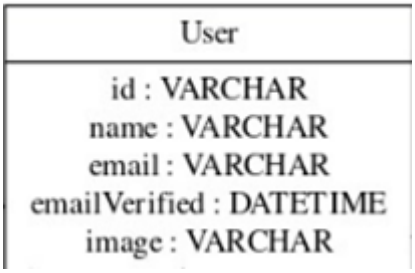
User to Game: One-to-many (one user can play many games)

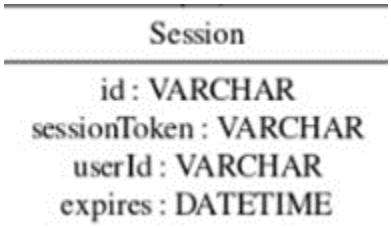
User to Question: One-to-many (one user can answer many questions)

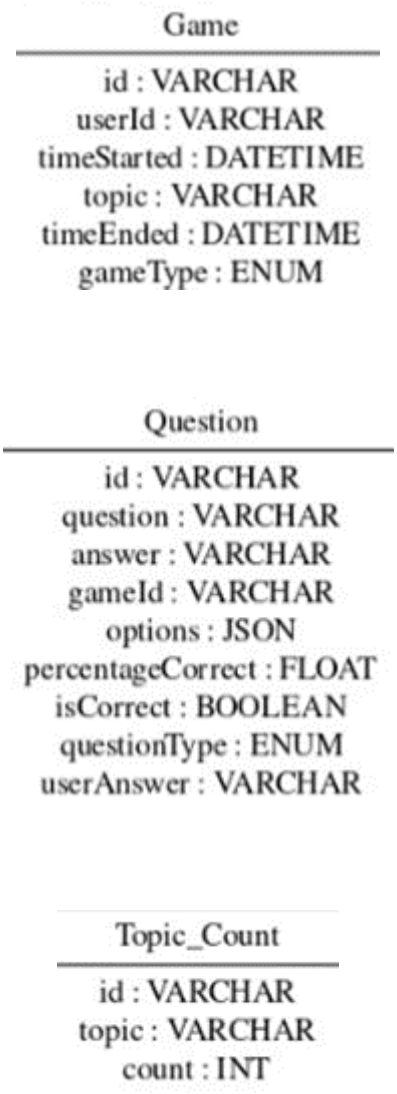
Game to Question: many-to-many (many games can have many questions)

Game to TopicCount: many-to-one (many games have one topic)

Database Schema:






Game
id : VARCHAR
userId : VARCHAR
timeStarted : DATETIME
topic : VARCHAR
timeEnded : DATETIME
gameType : ENUM
Question
id : VARCHAR
question : VARCHAR
answer : VARCHAR
gameId : VARCHAR
options : JSON
percentageCorrect : FLOAT
isCorrect : BOOLEAN
questionType : ENUM
userAnswer : VARCHAR
Topic_Count
id : VARCHAR
topic : VARCHAR
count : INT

Normalization:

- First Normal Form (1NF): Data in a table is in 1NF if it contains only atomic (indivisible) values, and there are no repeating groups or arrays.
- Second Normal Form (2NF): A table is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the primary key; there are no partial dependencies.
- Third Normal Form (3NF): A table is in 3NF if it is in 2NF and all transitive dependencies have been removed; no non-prime attributes depend on other non-prime attributes.
- Boyce-Codd Normal Form (BCNF): Every determinant (candidate key) is a superkey; there are no non-trivial functional dependencies on any superkey, ensuring that the table is free of redundancy.

Account Table:

- 1NF (First Normal Form): Yes, it has only atomic values in each column.
- 2NF (Second Normal Form): Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF (Third Normal Form): Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF (Boyce-Codd Normal Form): Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

Session Table:

- 1NF: Yes, it has only atomic values in each column.
- 2NF: Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF: Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF: Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

User Table:

- 1NF: Yes, it has only atomic values in each column.
- 2NF: Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF: Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF: Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

Game Table:

- 1NF: Yes, it has only atomic values in each column.
- 2NF: Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF: Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF: Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

Topic Count Table:

- 1NF: Yes, it has only atomic values in each column.
- 2NF: Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF: Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF: Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

Question Table:

- 1NF: Yes, it has only atomic values in each column.
- 2NF: Yes, it is in 1NF, and all non-prime attributes are fully functionally dependent on the primary key.
- 3NF: Yes, it is in 2NF, and all transitive dependencies have been removed.
- BCNF: Yes, every determinant is a superkey, and there are no non-trivial functional dependencies on any superkey.

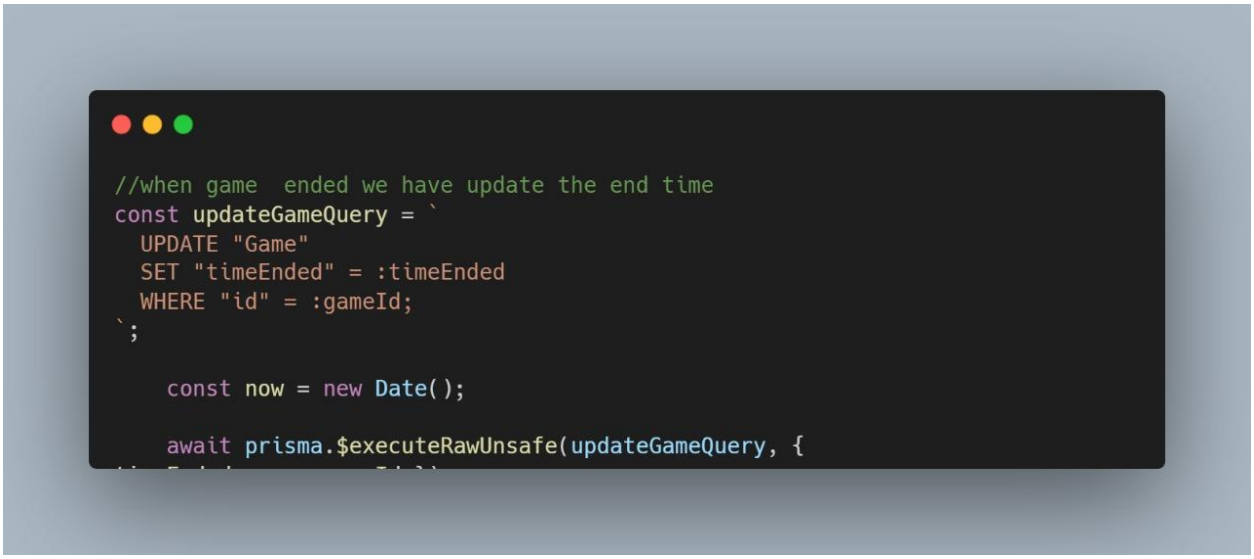
All the tables are in 1NF, 2NF, 3NF, and BCNF, indicating a well-normalized database design.

CODE:

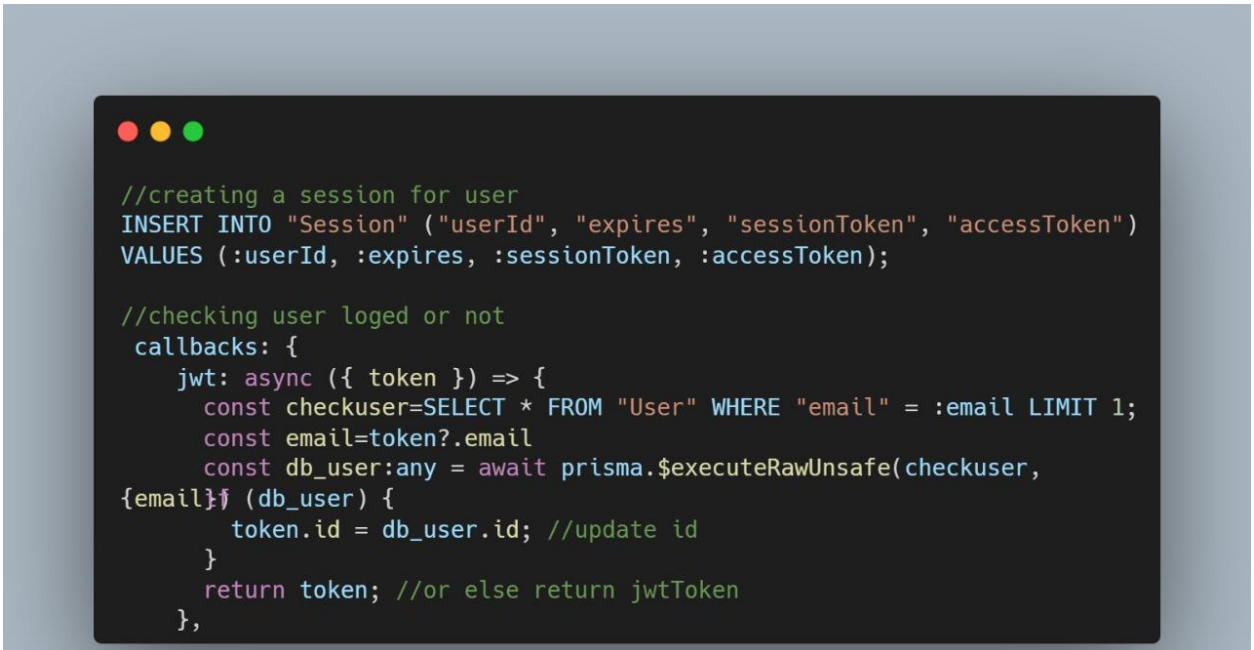
```
CREATE TABLE Account (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID())
  userId VARCHAR(255),
  type VARCHAR(255),
  provider VARCHAR(255),
  providerAccountId VARCHAR(255),
  refresh_token TEXT,
  access_token TEXT,
  expires_at INT,
  token_type VARCHAR(255),
  scope VARCHAR(255),
  id_token TEXT,
  session_state VARCHAR(255),
  FOREIGN KEY (userId) REFERENCES User(id) ON DELETE CASCADE,
  UNIQUE KEY unique_provider_providerAccountId (provider, providerAccountId),
  INDEX idx_userId (userId)
);
CREATE TABLE Session (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID()),
  sessionToken VARCHAR(255) UNIQUE,
  userId VARCHAR(255),
  expires DATETIME,
  FOREIGN KEY (userId) REFERENCES User(id) ON DELETE CASCADE,
```

```
  INDEX idx_userId (userId)
);
CREATE TABLE User (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID()),
  name VARCHAR(255),
  email VARCHAR(255) UNIQUE,
  emailVerified DATETIME,
  image VARCHAR(255),
  INDEX idx_games_userId (userId),
  INDEX idx_accounts_userId (userId),
  INDEX idx_sessions_userId (userId)
);
CREATE TABLE Game (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID()),
  userId VARCHAR(255),
  timeStarted DATETIME,
  topic VARCHAR(255),
  timeEnded DATETIME,
  gameType ENUM('mcq', 'open_ended'),
  FOREIGN KEY (userId) REFERENCES User(id) ON DELETE CASCADE,
  INDEX idx_userId (userId)
);

CREATE TABLE Topic_Count (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID()),
```

```
  topic VARCHAR(255) UNIQUE,
  count INT
);
CREATE TABLE Question (
  id VARCHAR(255) PRIMARY KEY DEFAULT (UUID()),
  question VARCHAR(255),
  answer VARCHAR(255),
  gameId VARCHAR(255),
  options JSON,  percentageCorrect FLOAT,
  isCorrect BOOLEAN,
  questionType ENUM('mcq', 'open_ended'),
  userAnswer VARCHAR(255),
  FOREIGN KEY (gameId) REFERENCES Game(id) ON DELETE CASCADE,
  INDEX idx_gameId (gameId)
);
```



```
//when game  ended we have update the end time
const updateGameQuery = `
  UPDATE "Game"
  SET "timeEnded" = :timeEnded
  WHERE "id" = :gameId;
`;

    const now = new Date();

    await prisma.$executeRawUnsafe(updateGameQuery, {
```



```
//creating a session for user
INSERT INTO "Session" ("userId", "expires", "sessionToken", "accessToken")
VALUES (:userId, :expires, :sessionToken, :accessToken);

//checking user loged or not
 callbacks: {
    jwt: async ({ token }) => {
      const checkuser=SELECT * FROM "User" WHERE "email" = :email LIMIT 1;
      const email=token?.email
      const db_user:any = await prisma.$executeRawUnsafe(checkuser,
{email});
      if (db_user) {
        token.id = db_user.id; //update id
      }
      return token; //or else return jwtToken
    },
```

```ts
//checking answer is correct

//finding question
const selectQuestionQuery = `
  SELECT * FROM "Question"
  WHERE "id" = :questionId;
`;

const question:any = await prisma.$executeRawUnsafe(selectQuestionQuery, { questionId });

//updating userinput
const updateQuestionQuery = `
  UPDATE "Question"
  SET "userAnswer" = :userInput
  WHERE "id" = :questionId;
`;

await prisma.$executeRawUnsafe(updateQuestionQuery, { userInput, questionId });

//updating user answer if question in mcq
const updateIsCorrectQuery = `
  UPDATE "Question"
  SET "isCorrect" = :isCorrect
  WHERE "id" = :questionId;
`;

await prisma.$executeRawUnsafe(updateIsCorrectQuery, { isCorrect, questionId });


//updating user answer if question in open-ended

const updatePercentageCorrectQuery = `
  UPDATE "Question"
  SET "percentageCorrect" = :percentageSimilar
  WHERE "id" = :questionId;
`;

await prisma.$executeRawUnsafe(updatePercentageCorrectQuery, { percentageSimilar, questionId
```

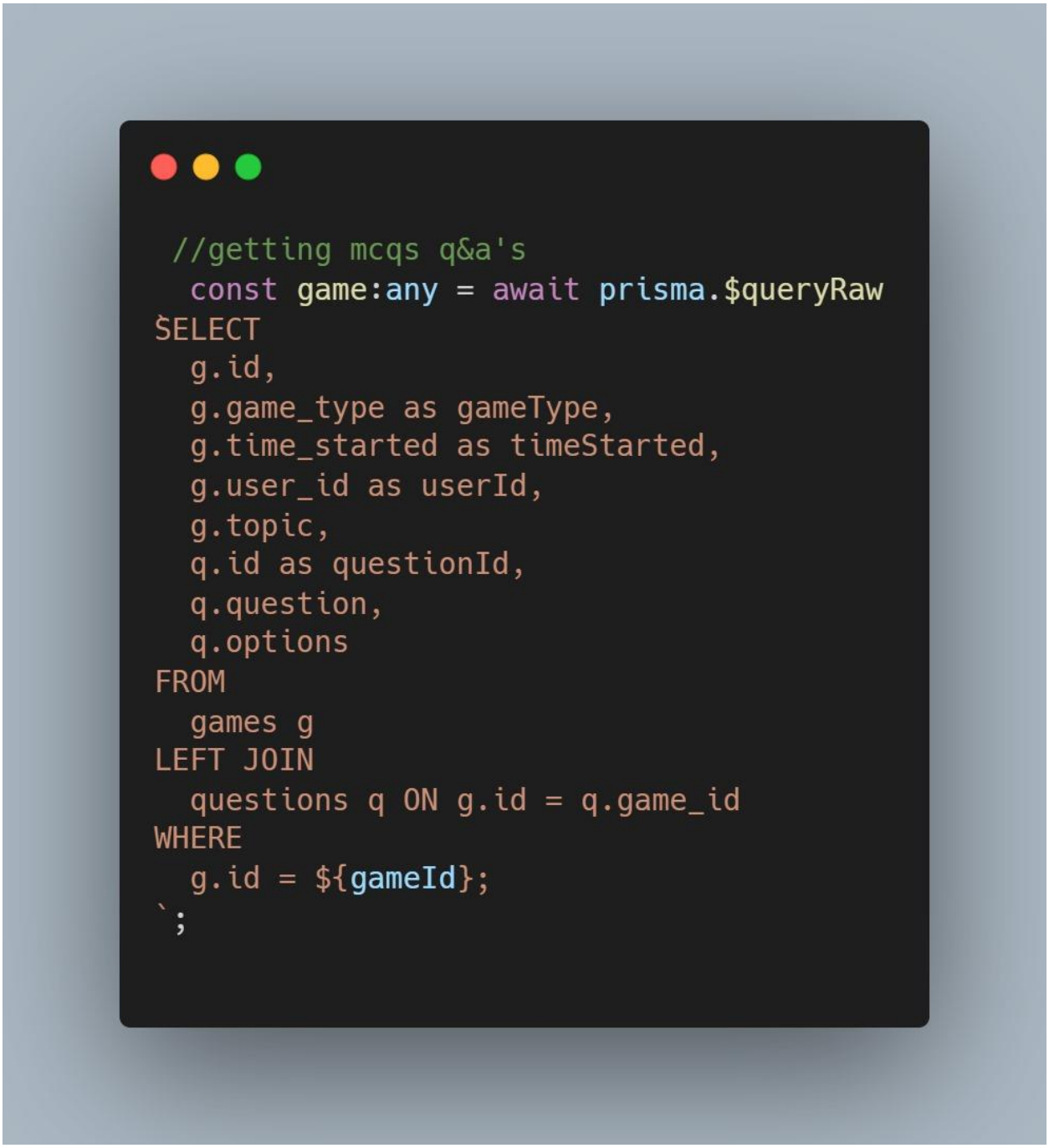
```ts
//getting few quizes attempted by the user to show in dashbaord
const selectGamesQuery = `
  SELECT * FROM "Game"
  WHERE "userId" = :userId
  ORDER BY "timeStarted" DESC
  LIMIT :limit;
`;

const games:any = await prisma.$executeRawUnsafe(selectGamesQuery, { userId, limit
```

```
// Creating the game in MySql.
    const insertGameQuery = `
      INSERT INTO games (game_type, time_started, user_id, topic)
      VALUES (:gameType, :timeStarted, :userId, :topic);
    `;

    const gameId = await prisma.$executeRawUnsafe(insertGameQuery,
{     gameType: type,
      timeStarted: new Date(),
      userId: session.user.id,
      topic,
    });
```

```
// Increment the count for the topic in to show in word cloud.
    const updateTopicCountQuery = `
    UPDATE topic_count
    SET count = count + 1
    WHERE topic = :topic
  `;

  await prisma.$executeRawUnsafe(updateTopicCountQuery, {
    topic,
  });
```

```
//inserting questions if open_ended
const insertOpenEndedQuestionsQuery = `
        INSERT INTO questions (question, answer, game_id, question_type)
        VALUES (:question, :answer, :gameId, 'open_ended');
      `;

      await prisma.$executeRawUnsafe(insertOpenEndedQuestionsQuery, questions.map((question) => ({
        question: question.question,
        answer: question.answer,
        gameId,
      })));
    }
```

```
//inserting questions if mcq
 const insertMCQQuestionsQuery = `
        INSERT INTO questions (question, answer, options, game_id, question_type)
        VALUES (:question, :answer, :options, :gameId, 'mcq');
      `;

      await prisma.$executeRawUnsafe(insertMCQQuestionsQuery, questions.map((question) =>
({       question: question.question,
        answer: question.answer,
        options: JSON.stringify([
          question.option1,
          question.option2,
          question.option3,
          question.answer,
        ].sort(() => Math.random() - 0.5)),
        gameId,
      })));
```



```
 //getting mcqs q&a's
  const game:any = await prisma.$queryRaw
`SELECT
  g.id,
  g.game_type as gameType,
  g.time_started as timeStarted,
  g.user_id as userId,
  g.topic,
  q.id as questionId,
  q.question,
  q.options
FROM
  games g
LEFT JOIN
  questions q ON g.id = q.game_id
WHERE
  g.id = ${gameId};
`;
```

```
//getting open_ended q&a's

 const gameQuery = prisma.$queryRaw`
 SELECT
   g.id,
   g.game_type as gameType,
   g.time_started as timeStarted,
   g.user_id as userId,
   g.topic,
   q.id as questionId,
   q.question,
   q.answer
 FROM
   games g
 LEFT JOIN
   questions q ON g.id = q.game_id
 WHERE
   g.id = ${gameId};
`;

 const game: any = await gameQuery;
```

```
//geting total game played by the user
  const gamesCountQuery:any = prisma.$queryRaw`
  SELECT COUNT(*) as count
  FROM games
  WHERE user_id = ${session.user.id};
`;

const games_count = await gamesCountQuery.count;
```

```
 //get the topics to show in hot topics
  const topicsQuery = prisma.$queryRaw`
  SELECT topic, count(*) as count
  FROM topic_count
  GROUP BY topic
  ORDER BY count DESC
  LIMIT 30;
`;

  const topics: any = await topicsQuery;
```

Limitations:

- In some cases, after repeatedly generating the quiz for a particular topic, Quizmify gives the same set of questions.
- In some instances, when generating a quiz, if we choose it to generate a greater number of questions, Quizmify takes a longer time to load and generate the quiz. In extreme cases, it fails to generate the quiz.
- Quizmify often generates questions that assess surface-level knowledge, such as facts or simple concepts.
- The Quizzes generated might not consider cultural or linguistic diversity, which can affect the appropriateness or fairness of questions for learners from different backgrounds.

References and URL's:

In building this project, we have referred to a few of the websites and resources that align with the objective and principle of our project. Below mentioned are the URL's:

https://www.quiz-maker.com/

https://www.mentimeter.com/features/quiz-presentations

https://www.flexiquiz.com/

https://quiz.com/

THANK YOU!!