

Rocket Trivia

CSS 475 Project - Final

Team Rocket

Bryan Castillo, Trece Wicklander, Fahad Alshehri, Fatih Ridha

March 11, 2018

Table of Contents

Team Rocket	5
Project: Rocket Trivia	5
Rocket Trivia Data	5
Entity: Group	5
Entity: Question	5
Entity: Answer	5
Entity: Category	5
Entity: Trivia Group	6
Entity: User	6
Entity: Game	6
Entity: Game Question	6
Entity: Game Question Answer	6
Use Case Categories	6
Trivia Question Maintainer	7
Trivia Game Player	7
Use Cases	7
Queries	8
Entity Relationship Diagram	10
Relational Model Schema	11
Constraints	12
Entity Constraints	12
Relationship Constraints	14
Design Decisions	16
Weak Entities vs. Strong Entities	16
GAME_QUESTION and GAME_QUESTION_ANSWER	16
Sign up and Authentication	16
User Administrators and Groups	17
Categories and Uniqueness	17
Tooling Assessment	18
Runtime Tools and Libraries	18
User Interface Layer	19
Vue.js	19
jQuery and Promise.js	19

Application Layer	19
Java and Java Virtual Machine	19
Spring Boot	19
Jetty Embedded HTTP Server	20
Spring REST Controllers	20
MyBatis	20
HikariCP	20
MySQL Connector/J	21
Database Layer	21
MySQL	21
Development Tools	21
Eclipse	21
Gradle	21
Postman	21
MySQL Workbench	21
Git and Gitlab	21
Hosting	22
AWS: EC2 and Linux	22
AWS: RDS	22
AWS: Route 53	22
AWS : Load Balancer	22
Database Implementation	23
Database Initialization	23
Accessing Database	23
Accessing Rocket Trivia Website	24
Schema	24
Constraints	25
Table Constraints	25
Test Data	26
Test Data: Method 1 - Manual Inserts	26
Test Data: Method 2 - Open Trivia	26
Test Data: Method 3 - Web Application	28
User Creation	28
Group Creation	29
Editing Group Content	29
Game Generation	32
Test Data: Method 4 - Web Application Play Simulator	33
SQL Query Statements	35

SQL Statements: User Creation and Sign In	36
SQL Statements: Group, Category, and Question Creation	36
SQL Statements: Game Creation	37
SQL Statements: Game Play	39
SQL Statements: Game Play - Find Games	40
SQL Statements: Game Play - Play Game	43
SQL Statements: Game Play - Game Review	45
SQL Statements: User Statistics	49
SQL Statements: Global Leaderboard	51
Normalization	53
USER	53
GROUP	53
CATEGORY	54
QUESTION	54
ANSWER	54
GAME	55
GAME_QUESTION	55
GAME_QUESTION_ANSWER	55
USER_GAMES	55
GROUP_ADMINISTRATORS	56
USER_GAME_QA_PICK	56
QUESTION_CATEGORIES	56
Normalized Schema	58
Project Evaluation and Reflection	59
Implementation Plan	60
Schedule	60
Areas of Work	61
Appendix	64
Schema Creation Statements	64
Schema and Users	64
Table Creation Statements	64
USER	64
GROUP	65
GROUP_ADMINISTRATOR	65
QUESTION	65
ANSWER	65
CATEGORY	65

QUESTION_CATEGORY	66
GAME	66
GAME_QUESTION	66
GAME_QUESTION_ANSWER	66
USER_GAMES	67
USER_GAME_QA_PICK	67
Trigger Creation Statements	67
USER	67
GROUP	68
QUESTION	69
ANSWER	70
CATEGORY	70
USER_GAMES	71
USER_GAME_QA_PICK	72
Basic Initial Data Population	72
Open Trivia Download And SQL Loading	76
Game Generation REST API / Java Code	81
Game Play Simulation	83

Team Rocket

Team Rocket is an engineering group with a focus on building consumer facing applications. Our members are:

- Bryan Castillo
- Trece Wicklander-Bryant
- Fahad Alshehri
- Fatih Ridha

Project: Rocket Trivia

Rocket Trivia is a database application for creating and running competitive trivia games. With Rocket Trivia you can challenge your friends and family to competitive trivia games.

Rocket Trivia Data

The Rocket Trivia application will need to store information in order to create, host, and view trivia games. The following data sets will be important for the application:

Entity: Group

A group in rocket trivia is a collection of users who manage and maintain a set of questions and categories of questions. The purpose of a Group is so that users can create their own questions to use in a game. While there may be a large public group of questions and categories to create games, a new small group could be created which targets a smaller set of people. For example, a small bar might create trivia questions about the establishment, the people working in the bar, and common customers.

Entity: Question

Trivia questions will be stored in the system.

Entity: Answer

Questions will need answers. For Rocket Trivia a question will need at least 1 correct answer and 1 or more incorrect answers.

Entity: Category

A trivia question may exist in 1 or more categories. Examples of categories might be science and sports.

Entity: Trivia Group

There may be multiple groups which manage trivia questions and categories. There will be at least 1 public group that any user may use questions for a trivia game, but users may want to create a group of questions and/or categories for hosting specialized games. An example of this might be a bar which hosts trivia night once a week after happy hour. The host could create their own group to host questions about the bar itself.

Entity: User

Information about users is important to Rocket Trivia. Certain users have the ability to manage trivia groups and questions. Knowing which users have rights to these groups and questions is important. Users participating in trivia games will be able to see how well they perform in certain areas.

Entity: Game

Rocket Trivia will allow certain users to host a competitive trivia game. Other users will be able to participate in the game itself. The game will need to have access to information such as:

- A name.
- A start time.
- The set of questions used in the game.
- The max time allowed for each question.
- The people currently participating in the game.
- The chosen answers for each participant and question in the game.

Entity: Game Question

When a game is created a random number of questions are samples from the set of questions which match the chosen categories selected when the game was created. The system may have 1000's of questions but a Game might only have 10 questions.

A Game Question is a chosen question used in a game. It includes a references to the question and the order that the question should appear in the game.

Entity: Game Question Answer

A question may be created with many correct and incorrect answers. When a game is created 1 random correct answer will be chosen from the known answers for a question and 3 incorrect answers will be chosen. This is done to ensure that all players in a game see the same possibilities.

Use Case Categories

Rocket Trivia has 2 main categories of use cases:

Trivia Question Maintainer

A trivia question maintainer creates and edits questions and categories within a given group. These users are the creators of content that could be used in a trivia game.

Trivia Game Player

A trivia game player creates and/or plays trivia games.

Use Cases

The following table describes the core use cases for Rocket Trivia and how those use cases interact with the entities in the design.

Name	Use Case Category	Description	Entity Interactions
User Signup	Maintainer or Player	A user creates a new account for Rocket Trivia.	A new User entity is created.
User Login	Maintainer or Player	A user logs in to Rocket Trivia.	The User entity is used to verify identity.
Create Group	Maintainer	A maintainer creates a group that will own questions, categories, and the associations between questions and categories.	A new Group is created and the user who created the group should become an administrator for the group.
Make Group Administrator	Maintainer	An administrator of a trivia group may designate additional users as administrators of the group.	New relationships between User and Group are created.
Create Category	Maintainer	An administrator for a trivia group creates a new question category.	A new Category is created with a relationship to the Group .
Create Question	Maintainer	An administrator for a trivia group creates a new question along with its associated correct answer and potential incorrect answers. The question is also associated with question categories.	A new Question is created along with the potential Answers . The relationships between the question and the Categories it belongs to are also created.

Create Game	Player	A player may create a game. The game will contain a number of randomly selected questions from the categories selected. The game will also have a maximum amount of time allowed per question and the end time for completing a game.	A new Game is created, a number of Game Questions are created based on the random selection, and Game Question Answers will be created from the selected potential answers. The correct answer for the given question will become a Game Question Answer and at least 3 incorrect answers will become Game Question Answers .
Join Game	Player	A player joins a trivia game. Multiple players may join and play a game. (<i>The game is meant to be competitive.</i>)	A new Plays relationship is created between a Game and a User . The start time for the relationship is set to the current date and time.
Answer Game Question	Player	A player chooses an answer for a question in a game.	A new Picks relationship is created between a Game Question Answer and a User . The relationship will include the date and time the Game Question Answer was chosen and the time taken for the user to make the choice.
View Game Statistics	Player	A player views game statistics. This shows information about the number of questions and the number correct for each player sorted by percent correct.	This is a query which uses Game , Game Question , Game Question Answer , Answer , User , and the Picks relationship.

Queries

Rocket Trivia will require several different queries depending on the use case or scenario.

Name	Query	Use Cases
Category List	List all question categories belonging to 1 or more specified trivia groups.	This will be used for creating trivia games and associating questions to categories.

Game Search	Find games which are accepting new participants.	This will be used to find games a user wants to participate in.
Get Next Unanswered Question	Retrieve the next question that should be answered in a game for a particular users.	This will be used when a user is playing a game.
Get Potential Answers	Get a list of possible answers for a question.	This will be used when a user is playing game. It should return 1 correct answer and multiple incorrect answers.
Get Game Scores	Get a list of participants and their current scores for a game.	This will be used to show a leaderboard for a game.
Get Question Statistics	Get statistics for questions such as the percentage of time a question is answered correctly and the average time participants take to answer a question.	The site will have a statistics page for the curious who want to see the most difficult questions or the questions that most people answer easily. This may also be used by users when they review how they did on a question compared to the rest of the world.
Get User Game Performance	Get the number of games a user has played in, the number of games won, the number of questions answered correctly, and the number of questions answered incorrectly.	This will be used on a user profile page showing how much they play and how well they do.
Get User Category Performance	Get the question performance of a user in specific question categories.	This will allow the user to see which categories they are best or worst in.

Entity Relationship Diagram

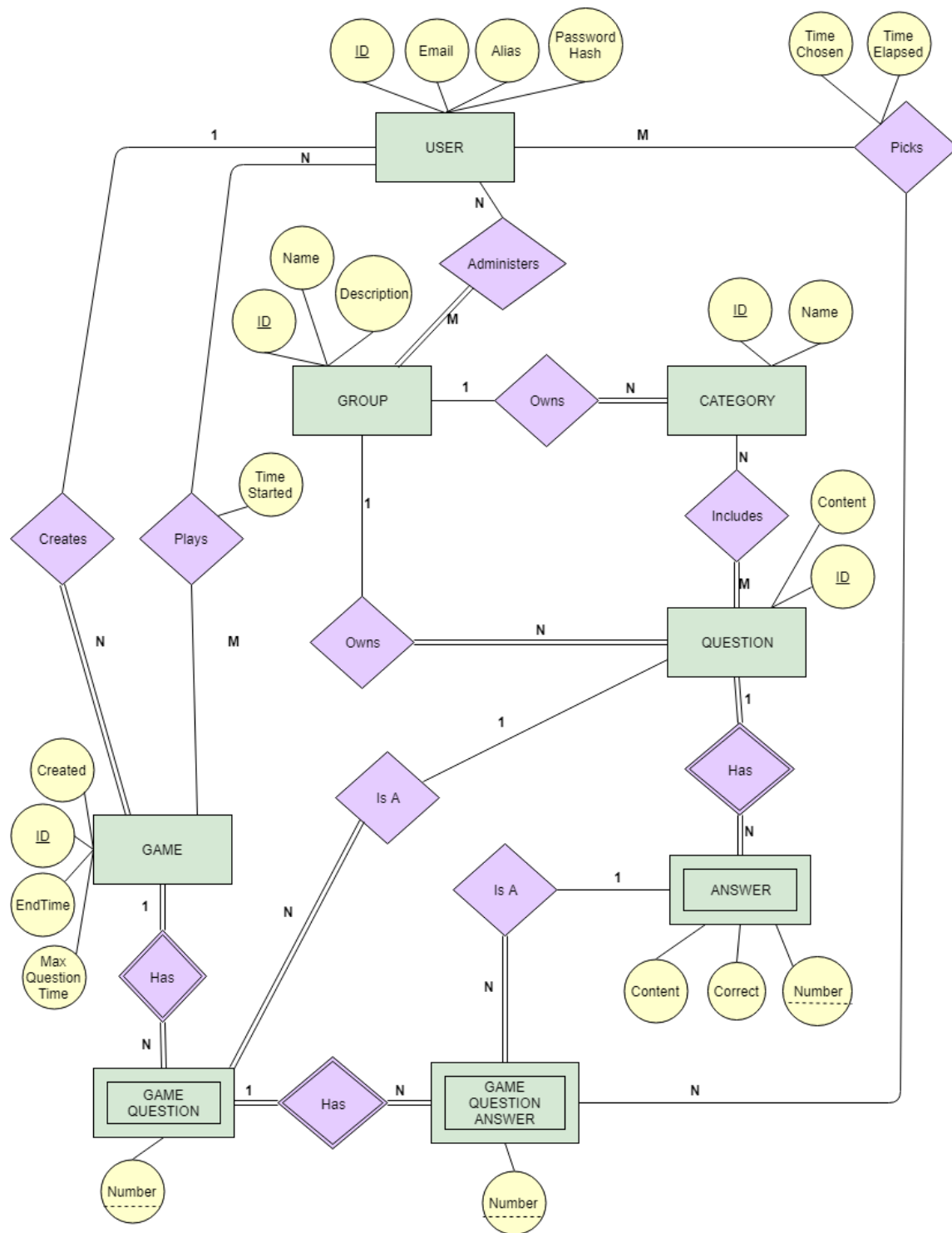


Figure 1: Rocket Trivia Entity Relationship diagram

Constraints

Entity Constraints

Relation	Attribute	Description	Constraints
User	Id	A unique number used to identify a user.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL. A non-negative integer.
User	Alias	A display name for the user.	A string that is 1 to 50 characters in length.
User	Email	The email address of the user. This is used to login.	A string that is 1 to 100 characters in length.
User	PasswordHash	A hex-encoded cryptographic hash (SHA-256) of the users password used for authentication.	A non-empty string 64 characters long. Each character must be a base 16 digit. 0-9 or A-F.
Group	GroupID	A unique number used to identify a group.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL. A non-negative integer.
Group	Name	A display name for a group.	A string that is 1 to 100 characters in length.
Category	CategoryID	A unique number used to identify a question.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL. A non-negative integer.
Category	CategoryName	The display name of a category.	A string that is 1 to 100 characters in length.
Category	GroupID	The group of the category.	[Referential Integrity] GroupID is a foreign key referencing the GroupID attribute of Group.
Question	QuestionID	A unique number used to identify a question.	Unique Key / Primary Key Cannot be NULL. A non-negative integer.
Question	Content	The text of the question.	A string that is 1 to 500 characters in length.

Question	GroupID	The owning group of the question.	[Referential Integrity] GroupID is a foreign key referencing the GroupID attribute of Group.
Answer	AnswerNumber, QuID	The primary key for an answer.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
Answer	AnswerNumber	A number used to identify an answer	A non-negative integer. [Entity Integrity] Cannot be NULL.
Answer	QuID	The question that owns the answer.	[Referential Integrity] QuID is a foreign key referencing QuestionID on Question.
Answer	Correct	A value used to indicate if the answer is correct or not. The value 1 is for true and the value 0 for false.	A small integer which must be 1 or 0. Cannot be NULL.
Game	GameID	A unique number used to identify a game.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL. A non-negative integer.
Game	Created	The date and time that the game was created.	A valid date and time. Cannot be NULL.
Game	EndTime	The date and time that the game will automatically close.	A valid date and time. Cannot be NULL.
Game	MaxQuestionTime	The maximum number of seconds that a player has to answer each question.	A non-negative integer. Cannot be NULL.
Game	UserID	The user who created the game.	[Referential Integrity] UserID is a foreign key referencing the UserID attribute of User.
Game_Question	Game_QNumber, GameID	The unique identifiers for a Game Question. Game_QNumber also determines the order of the questions in a game.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
Game_Question	GameID	The Game that the Question is part of.	[Referential Integrity] The GameID attribute references GameID on Game.
Game_Question	QuestionID	The question referred to	[Referential Integrity] The

tion		be the game question.	QuestionID is a foreign key referencing QuestionID on Question.
Game_Question_Answer	GameQANumber, GameQNum, GameID	The unique identifier for the game question answer.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
Game_Question_Answer	GameQNum, GameID	The game question the game question answer is for.	[Referential Integrity] The GameQNum and GameID attributes are a foreign key referencing the GameQNumber and GameID of GAME_QUESTION.
Game_Question_Answer	QuestionID, AnswerNumber	A references to the answer for the game question answer.	[Referential Integrity] The QuestionID and AnswerNumber attributes are a foreign key referencing QuestionID and AnswerNumber from Answer.

Relationship Constraints

Relation	Attribute	Description	Constraints
USER_GAMES	UserID, GameID	The identifier for a user game.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
USER_GAMES	UserID	The user of the user game.	[Referential Integrity] The UserID attribute references the UserID attribute of User.
USER_GAMES	TimeStarted	The time the user started or joined a game.	A valid date and time.
GROUP_ADMINISTRATORS	UserID, GroupID	The unique key which links the users who are administrators for a given group.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
GROUP_ADMINISTRATORS	UserID	The user that is an administrator.	[Referential Integrity] The UserID attribute references the UserID attribute of User.
GROUP_ADMINISTRATORS	GroupID	The group the user is administrator of.	[Referential Integrity] The GroupID attribute references the GroupID attribute of Group.

USER_GAME_QA_PICK	UserID, GameID, GameQNum, GameQANum	The unique identifiers for a user game question answer pick. This identifies the answer chosen by a user for a question in a game.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
USER_GAME_QA_PICK	GameID, GameQNum, GameQANum	The question answer in a game chosen by the user.	[Referential Integrity] The GameID, GameQNum, and GameQANum attributes are foreign keys which reference GameQANumber, GameQNum, and GameID on GAME_QUESTION_ANSWER.
USER_GAME_QA_PICK	UserID	The use which chose a game question answer.	[Referential Integrity] UserID is a foreign key which references UserID on USER.
USER_GAME_QA_PICK	TimeChosen	The date and time the user chosen the game question answer.	A valid date and time.
USER_GAME_QA_PICK	TimeElapsed	The number of milliseconds it took the player to pick the game question answer.	A non-negative integer.
QUESTION_CATEGORIES	CategoryID, QuestionID	The unique key that links questions to categories.	Unique Key / Primary Key [Entity Integrity] Cannot be NULL.
QUESTION_CATEGORIES	CategoryID	The category that a question is part of.	[Referential Integrity] CategoryID is a foreign key which references CategoryID on Category.
QUESTION_CATEGORIES	QuestionID	The question which is in some categories.	[Referential Integrity] QuestionID is a foreign key which references a QuestionID on Question.

Design Decisions

Weak Entities vs. Strong Entities

There are several 1 to N relationships in the database design. For some of these such as Category and Question, we used strong entities. We used weak entities for ANSWER, GAME_QUESTION, and GAME_QUESTION_ANSWER. We had debates about this. Making QUESTION and CATEGORY weak entities owned by GROUP made the relational model very complex. We considered making everything a strong entity using arbitrary unique identifiers to reduce the duplication of attributes. We felt that answers and game questions were more strongly associated with their owners than the other entities, so we left them as weak entities. However, we are unsatisfied with how complex the schema is for USER_GAME_QA_PICK. This could be reduced down to 2 attributes and the intended associate would be more clear from the schema. The team is still discussing this as a potential design change.

GAME_QUESTION and GAME_QUESTION_ANSWER

We chose to think of game questions and game question answers and entities themselves. We considered them at first as standard relationships in the ER diagram. We thought making them first class entities made the design more clear.

One potential issue with the design is that GAME_QUESTION and GAME_QUESTION_ANSWER reference QUESTION, and ANSWER respectively. We were thinking it would be possible to edit the text of a question or answer, but that may corrupt that information that is stored about the choice a user made for a question in a game. A potential design change we have began speaking about is making questions and answers immutable. Instead of changing them in place, we may have new questions or answers created and then mark old questions as inactive. We think that is too complex for our first iteration of the application though and will most likely not support question and answer edits.

Sign up and Authentication

We put some consideration into how users would sign up for Rocket Trivia. We needed users to be able to authenticate themselves so that we could control access to the creation of categories and questions within a group. We chose to use email as a login id and a password for authentication. We considered integrating with other services such as Google or Facebook authentication, but thought the integration would be complex. For passwords we chose not to store the actual password, but rather a hash of the chosen password using SHA-256. This was done so that if our data is leaked no passwords can be leaked.

We also considered an entity for storing a sign up request that required email verification prior to creating a user account. We skipped on that for the first round of the application because we didn't want to spend much time integrating with email providers.

User Administrators and Groups

In our design, the only relationship between a user and a group is that users can choose to administer a group to create categories and questions. This means that all members within a group are administrators and can create their own categories and questions as part of the group. This is to simplify the use cases for users and groups.

Categories and Uniqueness

We decided that categories will have a unique identifier separate from its name. The design allows for the same category name to be created more than once if the name is owned by other groups. We did this so that no individual group of users could claim ownership of an entire topic. For example, each group should have the ability to create their own science questions.

Tooling Assessment

The team has made an assessment of the tools, libraries, and application hosting for the Rocket Trivia application.

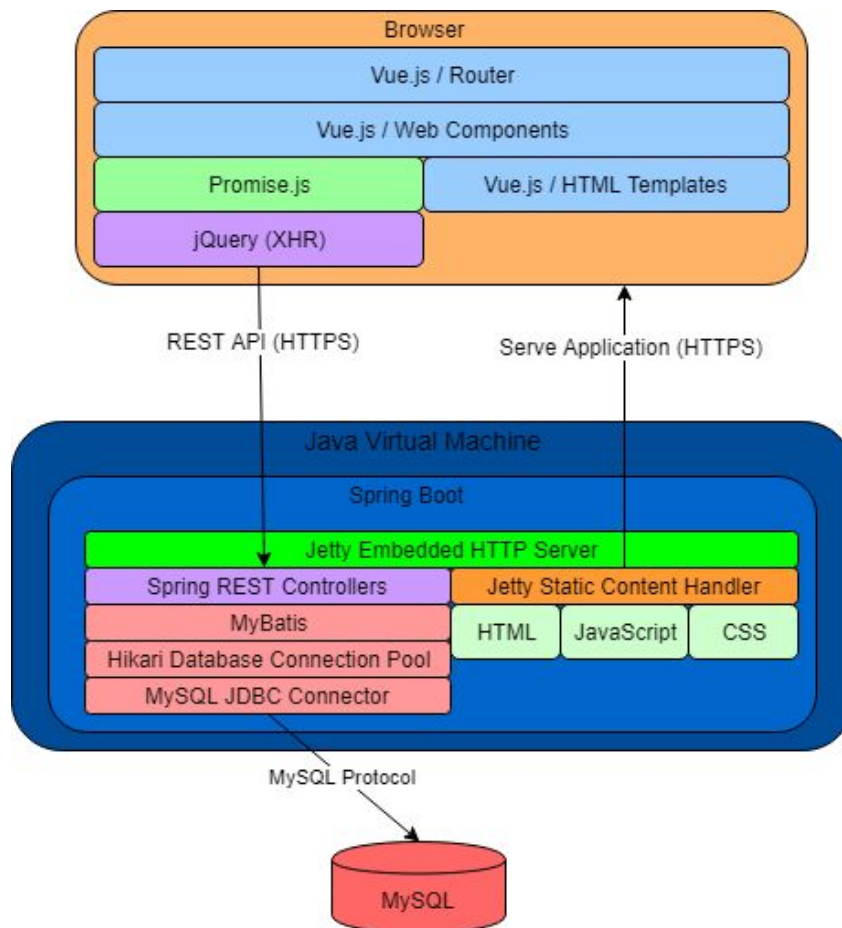
Runtime Tools and Libraries

The Rocket Trivia application is split into 3 layers:

1. Persistence or database layer
2. Application layer
3. The user interface layer

Each layer of the application is responsible for different portions of the application. The layers help the team develop features by properly separating responsibilities.

(Runtime Stack)



User Interface Layer

The user interface layer is responsible for interacting with users. The user interface is a web based application which users load and run with their web browser. The user interface uses the following tools:

Vue.js

We chose Vue.js for creating the user interface. Vue.js is a JavaScript framework for building interactive applications. Vue.js provides routing, component authoring, form binding, and template rendering. The library was chosen over other similar libraries such as Angular and React due to its ability to integrate easily with other technologies and development environments. The development stack allows engineers in Team Rocket to edit and run both the application layer code and user interface code using the same IDE and build tools.

jQuery and Promise.js

We chose to use jQuery and Promise.js for interaction with server-side REST APIs. jQuery is a larger library with functions for manipulating the user interface. Those portions of the library are not used through. jQuery has a well tested and highly configurable AJAX or XHR module for making HTTP requests and parsing JSON content. Promise.js was used to wrap jQuery HTTP responses. This allows the engineering team to mock out and cache requests without having to change user interface component code.

Application Layer

The Rocket Trivia application layer is responsible for:

1. Serving static content
2. Handling REST API requests
3. Authentication and authorization
4. Interacting with the database

Java and Java Virtual Machine

Our engineers chose to use Java as the primary language for developing the application layer. Most people on the team had experience with Java, the VM runs on most operating systems, there were multiple options for cloud hosting, there are connectors with a standard API abstraction for accessing relational databases, and there are a large number of libraries and development tools for the language.

Spring Boot

Spring boot is a container for running applications on a Java Virtual Machine. It has plugins for embedding an HTTP server, exposing REST APIs, and for exposing database connections to application code. The library provides flexible mechanisms for configuring applications for different environments and helps hook applications together using dependency injection. One of the other reasons Spring Boot

was chosen was that the server startup and boot sequence can detect the type of environment it is being run on which allowed the team to use the library to run the application directly from an IDE and run the application on common cloud hosting platforms.

Jetty Embedded HTTP Server

Jetty was chosen as the HTTP server for the application. The server is started as a plugin from Spring Boot. It could be used without writing explicit code and enabled the application to be hosted locally, on an IDE, or on various cloud deployment offerings. The server supports static content, dynamic handlers, and SSL.

Spring REST Controllers

Spring REST controllers were used for implementing web APIs. Spring REST controllers allow our engineers to quickly create entry points that the user interface layer can call easily using AJAX and JSON. The controllers also integrate with Spring Boot and dependency injection allowing the controllers to access lower level configured components. The REST controllers also provide a layer allowing the team to handle authentication and authorization before code interacts with the database.

MyBatis

MyBatis is a Java library that makes interaction between SQL and Java simpler while still allowing developers to write SQL directly. It eliminates much of the boilerplate code used to interact with JDBC connectors. The library does this by binding SQL statements to Java methods. At runtime the library will create implementations of the methods bound to the statements.

The following Java interface shows an example of how engineers would bind SQL with MyBatis to Java methods. MyBatis will generate an instance of the interface. When the methods are called they will be automatically bound to the given SQL statements.

```
public interface UserMapper {

    @Select("select * from User where email = #{0}")
    public UserDetail getUserDetailByEmail(String email);

    @Insert({
        "insert into User (Email, Alias, PasswordHash)",
        "values (#{0}, #{1}, #{2})"})
    public void createUserDetail(
        String email, String alias, String passwordHash);
}
```

HikariCP

HikariCP is a database connection pooling library for Java. It allows for applications to reuse database connections across requests. This helps increase performance by cutting down the overhead of establishing remote database connections. The library works with most JDBC compliant database drivers.

MySQL Connector/J

MySQL Connector/J is a JDBC compliant database driver for accessing the MySQL database from Java.

Database Layer

MySQL

The team choose MySQL as the database engine for Rocket Trivia. The database is supported on several operating systems and is free. This makes local development easy for the team. MySQL is also well supported by various cloud hosting providers such as AWS and Azure.

Development Tools

Team Rocket chosen the following tools for development and collaboration.

Eclipse

Eclipse was used as an IDE for developing the application. Eclipse has support for Java, JavaScript, HTML, and CSS editing and syntax highlighting. It integrated well with Spring Boot and Jetty allowing interactive debugging of application layer code for embedded web servers.

Gradle

Gradle is a Java based build tool. It is used for automating library dependency management, compilation, and build artifacts. The tools helps with both setting up the development environment and deploying the code to a cloud hosted environment.

Postman

Postman is an interactive test tool used for testing REST APIs. It allows engineers on the team to build and test APIs without having to write a user interface to invoke HTTP requests.

MySQL Workbench

MySQL Workbench as a user interface for viewing the structure of a MySQL database and for authoring and executing SQL statements. The tool makes it easy to develop SQL and test queries.

Git and Gitlab

The team is using git for managing source code. Gitlab is used as a central repository for sharing code. Gitlab offers free private repositories. This is important for the team because it allows the team to not worry about storing credentials within source controlled configuration files.

Hosting

The Rocket Trivia application is hosted on Amazon Web Services. More people had experience with AWS than other cloud hosting providers such as Azure. AWS also provides free service tiers for development purposes.

AWS: EC2 and Linux

The application layer is hosted on AWS using EC2 and Amazon's custom Linux image. A custom Linux server provided the ability for the team to easily deploy code directly from Gitlab. Custom scripts are used to pull the code from git and Gradle is used to build the binary from source directly on the host. The custom Linux image allows engineers to login to the host via SSH and view logs.

AWS: RDS

Amazon Relational Database Service is used for hosting MySQL. RDS provides tools for easily creating and managing MySQL instances.

AWS: Route 53

Route 53 is a DNS service that works well with Amazon web services. It allows the team to have a friendly URL which is directed to the physical application hosts.

AWS : Load Balancer

The AWS Load Balancer service is used as the first server layer for the application. This was chosen because it supports integration with Amazon SSL certificate management, which allows the Rocket Trivia application to be run using HTTPS. This is important since the application requires passwords for authentication.

Database Implementation

Database Initialization

The database initialization was performed by taking our Relational Model and transforming it into creation statements in MySQL. A series of scripts and randomization tools were used to populate the database. Each database consists of at least 2000 tuples for different questions to play the trivia game.

Accessing Database

The database is a MySQL instance hosted on AWS. The database may be accessed through a mysql command line client or a tool such as MySQL Workbench. A test account has been created for instructors and graders. Here is the connection information and credentials for accessing the database.

Hostname:	rocket-trivia-db.cefd0wwguptt.us-east-1.rds.amazonaws.com
Port:	3306
Username:	css451-tester
Password:	cuniculus
Default Schema:	rocket

Here is an example using the connection info about to connect to the database using the MySQL command line client.

```
C:\bryanc\education\CSS475\rocket\rocket-maintenance>mysql
--host=rocket-trivia-db.cefd0wwguptt.us-east-1.rds.amazonaws.com --user=css451-tester
--password=cuniculus rocket
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 35708
Server version: 5.6.37-log MySQL Community Server (GPL)
```

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql> show tables;
+-----+
| Tables_in_rocket |
+-----+
| Answer            |
| Category          |
| Game              |
| Game_Question     |
```



```

| Game_Question_Answer |
| Group                |
| Group_Administrator  |
| Question              |
| Question_Category    |
| User                  |
| User_Game_QA_Pick    |
| User_Games           |
+-----+
12 rows in set (0.11 sec)

mysql>

```

Accessing Rocket Trivia Website

Portions of the Rocket Trivia website are up and running. By using the application, the database may be manipulated. The website can be found at the given URL:

<https://www.uwbrockettrivia.net>

A test user has been created for graders and instructors to access. Here are the credentials:

Email: css451-tester@uw.edu
Password: cuniculus

While the application supports the dynamic creation of a user that graders and instructors may use, this test user has administrative privileges to trivia groups already loaded with categories and questions.







Remote access to our database is available to anyone and they may create a new user at the following URL:

<https://www.uwbrockettrivia.net/#/user/create>

Anyone can create questions and list which category this question should be under. Feel free to sign in using the previous sign-in information to test, create, or play our trivia game.

Schema

We created SQL scripts for generating the schema for the Rocket Trivia database. These scripts were targeted for MySQL 5.6 and 5.7. The scripts were checked into our source repository. We used multiple SQL files for the schema to make it easier to review our schema. We used a file naming format of <NN>.<purpose>.sql. The numbers are used to specify which order the files should be applied in. This makes it easy for us to update our schema incrementally.

 01.create_schema_and_db_users.sql	2/27/2018 7:50 PM	SQL Text File	1 KB
 02.user_table.sql	2/28/2018 8:33 PM	SQL Text File	2 KB
 03.groups.sql	2/28/2018 9:19 PM	SQL Text File	2 KB
 04.question_answer_category.sql	3/1/2018 7:26 AM	SQL Text File	4 KB
 05.game_question_answer.sql	2/24/2018 1:26 PM	SQL Text File	2 KB
 06.user_game.sql	3/1/2018 7:30 AM	SQL Text File	3 KB

The first file contains the statements for generating a logical database, users, and grants for the rocket database. This first file should be run as the root user. The following files should be run as the rocket user. They generate the tables.

All SQL statements found in these files are also listed in the Appendix.

Constraints

We implemented constraints and various verifications in multiple ways for our database.

Table Constraints

We implemented basic constraints on our tables for uniqueness, primary keys, and foreign keys using standard SQL syntax.

Initially we attempted to add check constraints to our tables as well. Here is an example of one such constraint.

```
ALTER TABLE User
ADD CONSTRAINT Email_Check
CHECK (CHAR_LENGTH(Email) >= 1 AND NOT Email REGEXP '^[:space:]' AND NOT Email REGEXP '[:space:]$');
```

MySQL accepted these statements without giving an error. We had assumed that the constraints were being respected. We later found out that MySQL 5.x does not support check constraints even though it parses the SQL when run.

We still wanted to add checks at the database level. We implemented our constraints using triggers. See the Appendix for the full list of triggers we created. The following is an example showing tests to make sure that our triggers will generate errors on empty strings or strings beginning with whitespace for the email and alias attributes on the User table.

```
mysql> insert into User (Email, Alias, PasswordHash) values ('', 'bob', 'XXXXXX');
ERROR 1644 (45000): Email Error
```

```
mysql> update User set Alias = ' bryan' where Alias = 'bryan';
ERROR 1644 (45000): Alias Error
```

Test Data

The team took multiple approaches to populate test data for the Rocket Trivia database. The approach depended on the complexity of the data, the utility of the data, and the difficulty in generating data.

Before describing the methods for generating test data, it is useful to categorize the types of data in the database. The tables can be classified into 3 areas:

- User information for authentication.
- Question content information.
- Game play information.

The first type of information was basic user data. There is only 1 main table for the list of users. The 2nd type of information is about the questions that can be used in a game. The 3rd type of data is information about people playing trivia games.

Test Data: Method 1 - Manual Inserts

The first method the team used to create test data was manual inserts. The initial data loads can be found in the appendix section titled “Basic Initial Data Population”. There were 2 issues that were somewhat tricky for the initial data loads.

The first was that passwords had to be translated into hashes. This was done with a short python script also listed in the appendix. The second issue was figuring out how to deal with auto generated primary key values. For questions and categories we used identity columns for MySQL. After inserting a question we had to retrieve the last id generated and use that when creating a question. We found that MySQL supports variable setting and a function to get the last id.

Test Data: Method 2 - Open Trivia

The team realized that we needed a larger number of questions to load into the database in order to generate test game play. Manually creating questions was quite painful. We found a website which opentb.com which contains a large number of trivia questions and answers. The site offered an API for searching and retrieving the questions.

We wrote 2 python scripts to pull the data from the website. The scripts can be found in the appendix under “Open Trivia Download And SQL Loading”. The first script uses the HTTP API provided by opentb.com to pull trivia questions and save them to disk in JSON format.

Here is an example of one of the questions pulled from opentb.com.

```
{
  "category": "Geography",
```

```

    "correct_answer": "Washington D.C",
    "difficulty": "easy",
    "incorrect_answers": [
        "Seattle",
        "Albany",
        "Los Angeles"
    ],
    "question": "Which city is the capital of the United States of America?",
    "type": "multiple"
},

```

The information provided by opentb.com matched closely with the format of questions and answers used by rocket trivia. The job of the 2nd script was to translate the JSON data into SQL statements which could be run directly against the database.

The second script read all of open trivia's questions into memory and ran through the following logic:

1. It would filter out any questions which did not have at least 3 answers.
2. It would filter out any questions where the question content or the answer content used characters beyond the basic ASCII character set. We did this because we had difficulty figuring out how to set up the database to handle full unicode.
3. We filtered out duplicate questions. The Open Trivia database API would sometimes return duplicate questions when paginating through results.
4. The questions would be grouped by category. Grouping by category made it easier to write the script to create the category once, lookup the new category id and use it to insert all questions in the given category.
5. The insert statements were written out for the questions and answers. The content had to be escaped to convert the statements into literal SQL strings.

The output of the second python script was a SQL script which we could then run against the database.

Here is a snippet of the SQL generated by the python script. This snippet shows the insertion of a category, 1 question with that category, and the answers for the question.

```

INSERT INTO Category (CategoryName, GroupId) VALUES ('Geography', @groupId);
SET @categoryId = LAST_INSERT_ID();

INSERT INTO Question (Content, GroupId) VALUES ('Which city is the capital of the United States of
America?', @groupId);
SET @questionId = LAST_INSERT_ID();
INSERT INTO Question_Category (CategoryId, QuestionId) VALUES(@categoryId, @questionId);
INSERT INTO Answer VALUES(1, @questionId, 'Washington D.C', 1);
INSERT INTO Answer VALUES(2, @questionId, 'Seattle', 0);
INSERT INTO Answer VALUES(3, @questionId, 'Albany', 0);
INSERT INTO Answer VALUES(4, @questionId, 'Los Angeles', 0);

```

We had a choice to have the second python script emit SQL or we could have had the python script use a database API to perform the inserts. Using a database API would have removed the need to escape strings, but the generated SQL file was also convenient because we could reload the data easily without rerunning the python scripts.

The generated SQL file contained over 15,000 SQL statements. One thing we noticed was that running 15,000 statements took a long time to reload the test data. We found out that by default each statement was running as its own transaction. Once we changed the script to turn off auto-commit we found that the data loaded much faster.

Test Data: Method 3 - Web Application

In addition to manually loading data by writing SQL statements and bulk loading from another trivia site, we built out portions of the web application to support the creation of users, questions, and games. This allows multiple members of the team to easily create question content for testing.

User Creation

The first feature we built in the app was the creation of users. The following screen can be used to generate a new user. After creating a user the results can be seen by querying the User table.

<https://www.uwbrockettrivia.net/#/user/create>

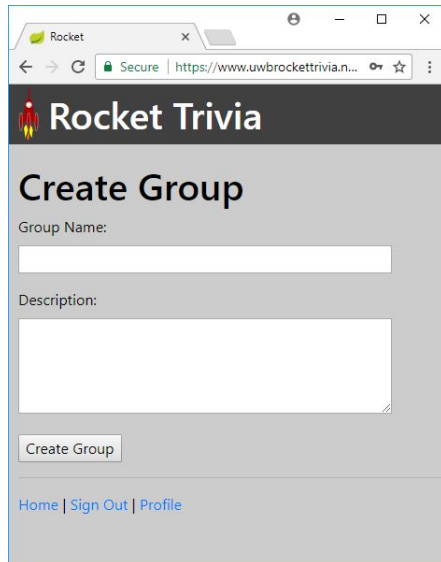


The screenshot shows a web browser window with the URL <https://www.uwbrockettrivia.net/#/user/create>. The page has a dark header with the "Rocket Trivia" logo and title. Below the header, the main heading is "Create Account". The form contains five input fields: "Alias:", "Email:", "Email (verify):", "Password:", and "Password (verify):". Each field is represented by a white rectangular input box. At the bottom of the form is a "Create Account" button. Below the button, there are two links: "Sign In" and "Create New User".

Group Creation

The next useful entity to create is a trivia group. One of these must be created to generate questions.

<https://www.uwbrockettrivia.net/#/groups/create>

A screenshot of a web browser window showing the 'Create Group' page for Rocket Trivia. The browser's address bar shows the URL 'https://www.uwbrockettrivia.net/#/groups/create'. The page has a dark header with the 'Rocket Trivia' logo. Below the header, the title 'Create Group' is displayed. There are two input fields: 'Group Name:' with a single-line text box, and 'Description:' with a multi-line text area. A 'Create Group' button is located below the description field. At the bottom of the page, there are links for 'Home', 'Sign Out', and 'Profile'.

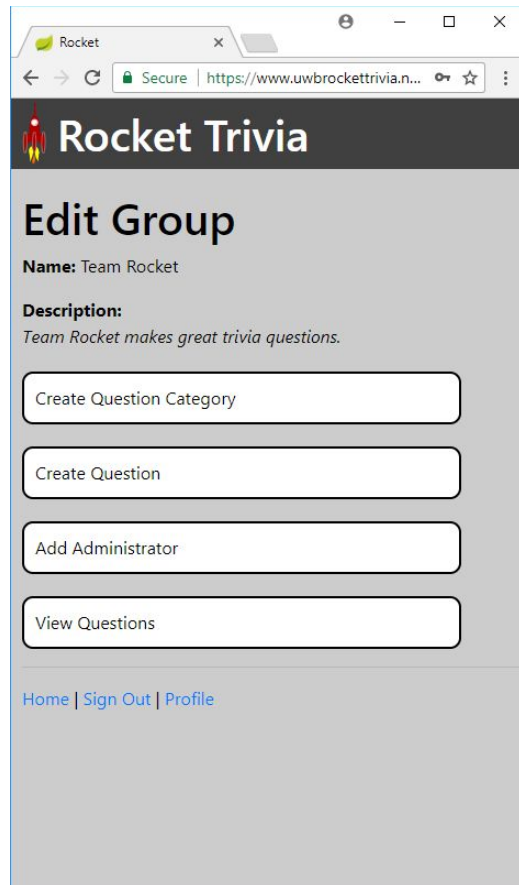
Editing Group Content

Once a trivia group has been created, content can be added to it. The following screen is used to enter an edit landing page for a group.

<https://www.uwbrockettrivia.net/#/home/creator>

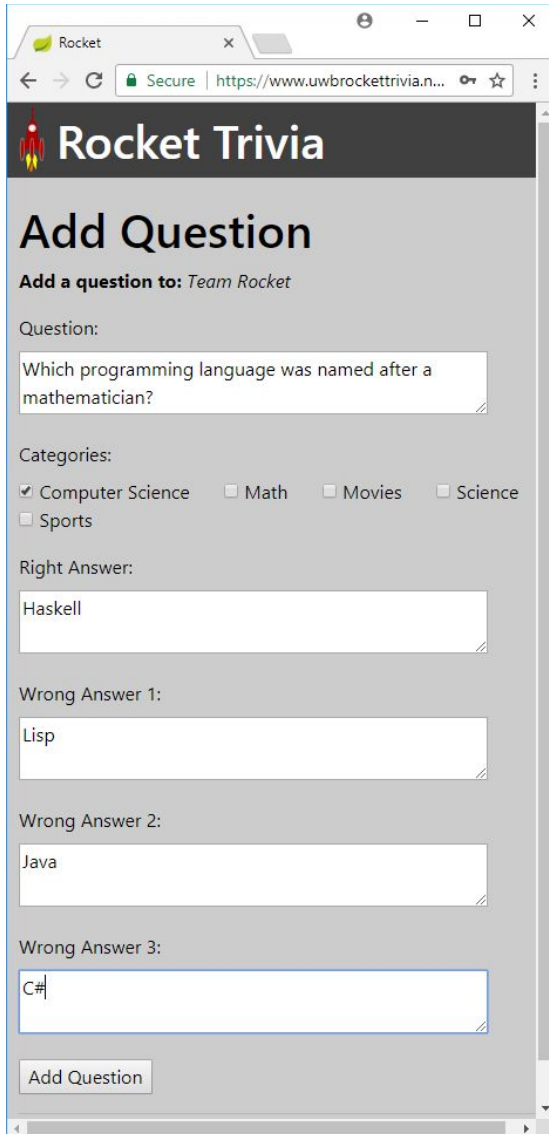


Once a group is chosen a screen will be shown with links to perform actions in the group to create a new question category, create a question, or add a new administrator to the group. All of these actions will edit the live database and were used to generate extra test data.



Example of adding a question:

<https://www.uwbrockettrivia.net/#/group/1/add-question>



The screenshot shows a web browser window with the title 'Rocket Trivia'. The URL bar shows 'https://www.uwbrockettrivia.net...'. The page has a dark header with a rocket icon and the text 'Rocket Trivia'. Below the header, the main heading is 'Add Question'. Underneath, it says 'Add a question to: Team Rocket'. The form includes a 'Question:' label followed by a text input field containing 'Which programming language was named after a mathematician?'. Below this is a 'Categories:' section with five checkboxes: 'Computer Science' (checked), 'Math', 'Movies', 'Science', and 'Sports'. The form also has four 'Wrong Answer' fields: 'Right Answer:' with 'Haskell', 'Wrong Answer 1:' with 'Lisp', 'Wrong Answer 2:' with 'Java', and 'Wrong Answer 3:' with 'C#'. At the bottom of the form is an 'Add Question' button.

Game Generation

The team found that game generation by hand was very tedious. There are multiple foreign keys which make data generation take a long time and it makes it easy to get the wrong ids. We built the application to support game generation.

The process of generating a game involves picking some game parameters such as the number of questions to sample and the categories to sample from. The intent is to pick questions randomly from database and put the question and answer in an order that will be presented to each user who plays the game.

The following screen shows an example of using the user interface to generate game data.

<https://www.uwbrockettrivia.net/#/games/create>

The screenshot shows a web browser window with the URL <https://www.uwbrockettrivia.net/#/games...>. The page title is "Rocket Trivia" with a rocket icon. The main heading is "Create A Game".

Form fields and options include:

- Number of questions:** A dropdown menu set to "20".
- Maximum time per question:** A dropdown menu set to "30 seconds".
- Game end date and time:** A text input field containing "03/31/2018 12:00 PM".
- Question Groups:** A list of checkboxes with counts:
 - ☒ Open Trivia (2393)
 - ☐ Avengers (8)
 - ☐ Team Rocket (4)
 - ☐ Best Group (1)
- Question Categories:** A list of checkboxes with counts:
 - ☐ Entertainment: Video Games (583)
 - ☒ Entertainment: Music (226)
 - ☒ History (182)
 - ☐ Geography (169)
 - ☐ General Knowledge (146)
 - ☐ Entertainment: Film (143)
 - ☐ Science & Nature (129)
 - ☐ Entertainment: Japanese Anime & Manga (114)
 - ☐ Entertainment: Television (105)
 - ☐ Science: Computers (97)
 - ☐ Entertainment: Cartoon & Animations (63)
 - ☐ Sports (62)
 - ☐ Entertainment: Books (61)
 - ☐ Vehicles (54)
 - ☐ Animals (36)
 - ☐ Celebrities (36)
 - ☐ Mythology (33)
 - ☐ Entertainment: Board Games (29)
 - ☐ Entertainment: Comics (29)
 - ☐ Politics (29)
 - ☐ Science: Mathematics (23)
 - ☐ Entertainment: Musicals & Theatres (19)
 - ☐ Science: Gadgets (15)
 - ☐ Art (10)

A "Create Game" button is located at the bottom of the form. At the very bottom of the page, there are links: [Home](#) | [Sign Out](#) | [Profile](#).

The game generation UI will call a REST API on the same http server which serves the UI. Snippets of code used to implement that API can be found in the appendix under "Game Generation REST API / Java Code".

Test Data: Method 4 - Web Application Play Simulator

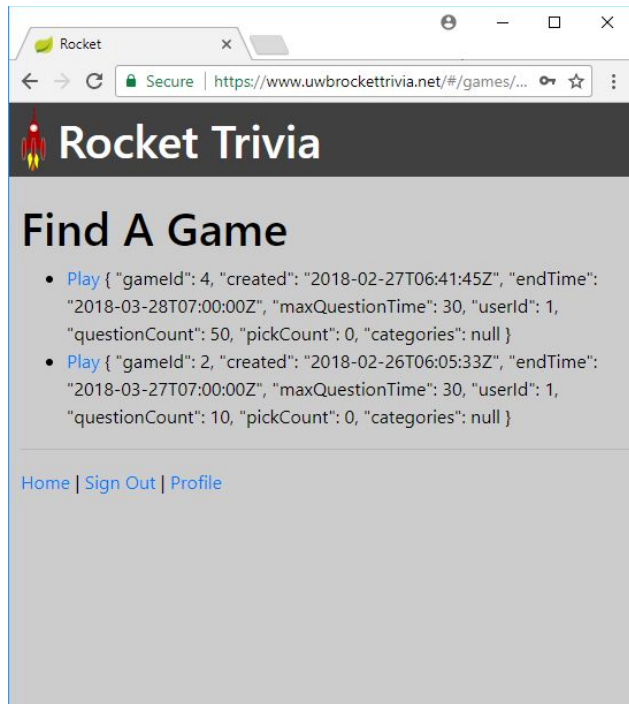
The last type of data we needed to populate for testing was actual game play. This would involve a user choosing to play a trivia game and going through answer the questions in a given game. We did not have enough time to create the full user interface to present a user each question. We decided to create a temporary feature in the application to simulate game play.

This includes a user interface and a REST API method we do not intend to keep, but it allowed us to generate data easily by reusing Java Mappers we had already created.

To simulate game play find existing games at the following page:

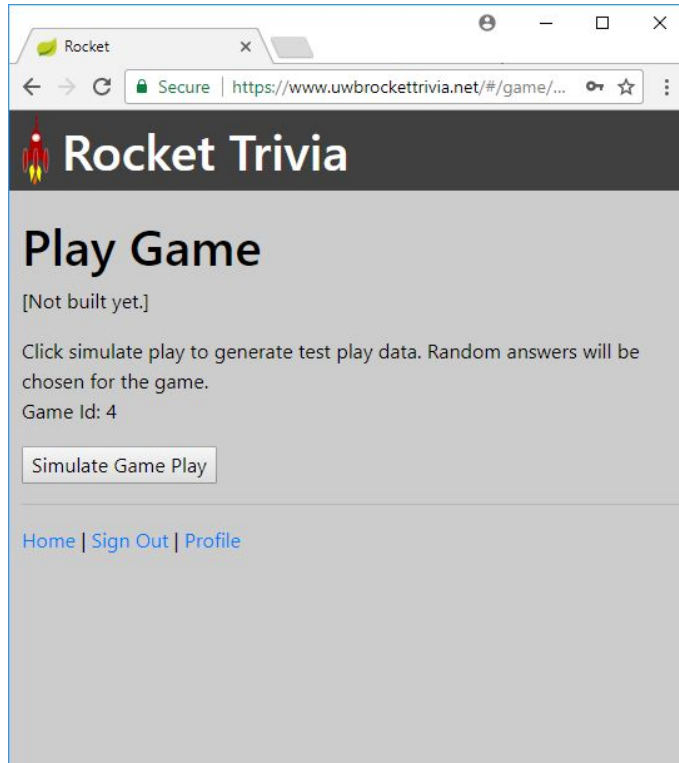
<https://www.uwbrockettrivia.net/#/games/find>

If there are no games you can create a new game as previously described.



Click the Play link will take you to a play page for that game.

<https://www.uwbrockettrivia.net/#/game/4/play>



Next click on “Simulate Game Play” to have your current user randomly answer all questions in the chosen game.

When the user interface simulates game play it does so by calling a custom REST API which picks random answers. Some of the code for game simulation can be found in the appendix under “Game Play Simulation”.

SQL Query Statements

Rocket Trivia required several SQL statements for the application. It required basic insert statements for creating data, basic select statements for simple data access, and complex queries to show statistics or for review of trivia game performance. The SQL statements are shown below are grouped by areas of the application. Not all SQL statements used by the application are listed in this document, but they can be found in the accompanying source code. All SQL statements are configured in Mapper files as previously described. Here is the listing of files which contain SQL used by the main application.

Directory of D:\bryanc\education\CSS475\rocket\rocket-service\src\main\java\rocket\mappers

03/11/2018	09:32 PM	2,369	CategoryMapper.java
03/11/2018	09:32 PM	316	DBInfoMapper.java
03/11/2018	09:32 PM	9,701	GameMapper.java
03/11/2018	09:32 PM	2,234	GroupMapper.java
03/11/2018	09:32 PM	3,324	QuestionMapper.java
03/11/2018	09:32 PM	1,472	TriviaStatsMapper.java

03/11/2018 09:32 PM
03/11/2018 09:32 PM

3,622 UserGameMapper.java
2,675 UserMapper.java

SQL Statements: User Creation and Sign In

The following statements were used for creating users and verifying their sign in credentials.

SQL Statement	Purpose
SELECT * FROM User WHERE email = #{0}	Used during sign in to look up a user and verify if the users password matches. (The password is stored as a SHA-256 hash.)
SELECT userId, email, alias FROM User WHERE email = #{0}	Used for general user lookup without returning a password hash. This was done so that password hashes are not returned from the server to the client.
INSERT INTO User (Email, Alias, PasswordHash) VALUES (#{0}, #{1}, #{2})	Adds a new user. The database assigns a new unique id afterwards.

SQL Statements: Group, Category, and Question Creation

Once users are created in the application, they may create new questions to be used in trivia games. Users may create a Trivia Group, Categories, and Questions.

Details about the creation process including screenshots of the user interface which uses these SQL statements can be seen in the section: "Test Data: Method 3 - Web Application".

SQL Statement	Purpose
INSERT INTO `Group` (GroupName, Description) VALUES (#{0}, #{1})	Creates a new Group.
INSERT INTO Group_Administrator (GroupId, UserId) VALUES (#{0}, #{1})	Adds a user as an administrator for a group. Only administrators of a group may create content for that group. The user who initially create a group is automatically added as an administrator.

SELECT COUNT(*) > 0 AS isAdmin FROM Group_Administrator ga WHERE ga.UserId = #{0} and ga.GroupId = #{1}	Returns 1 or 0 for true or false indicating if a user is an admin for a group. This is done by the application before inserting information or updating information concerning a group.
INSERT INTO Category (GroupId, CategoryName) VALUES (#{0}, #{1})	Inserts a new category for questions within the group.
SELECT c.* FROM Category c WHERE c.GroupId = #{0} ORDER BY c.CategoryName	Gets the categories for a group. This is useful when displaying the UI for creating a question. The user is shown a checkbox indicating which categories the question will be part of.
INSERT INTO Question (Content, GroupId) VALUES (#{0}, #{1})	This is used to start the creation of a question. It adds the question with the given content to a group. The question is assigned a unique id.
SELECT LAST_INSERT_ID()	After inserting a question, the assigned id is retrieved with the given statement.
INSERT INTO Question_Category VALUES (#{0}, #{1})	After the id is known for a question, it is associated with a set of categories. At least 1 category must be assigned which is enforced by the application.
INSERT INTO Answer VALUES (#{0}, #{1}, #{2}, #{3})	Next, the answers for a question are inserted. At least 1 correct answer must be added, and at least 3 incorrect answers. This is enforced by the application.

SQL Statements: Game Creation

Trivia games are created by random sampling questions from chosen trivia groups and categories. A description of the process and screenshots which use the following queries can be found in section: "Test Data: Method 3 - Web Application".

SQL Statement	Purpose
---------------	---------

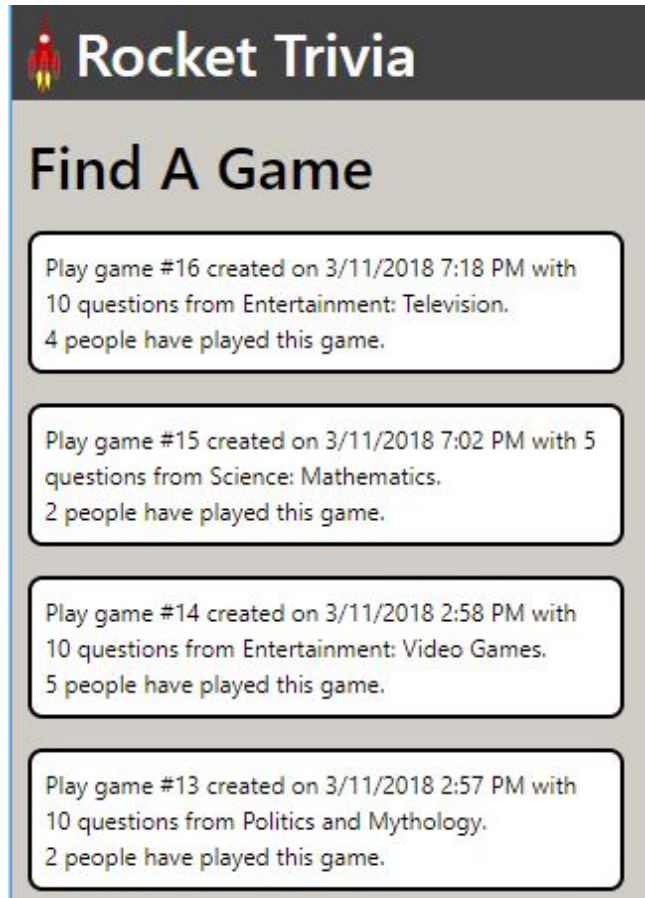
<pre> SELECT g.groupId, g.groupName, COUNT(*) as QuestionCount FROM Question q JOIN `Group` g ON (q.groupId = g.groupId) GROUP BY g.groupId, g.groupName HAVING COUNT(*) > 0 ORDER BY QuestionCount DESC, g.groupName </pre>	<p>This query is used to show trivia groups you may want to sample from when creating a game. It only shows groups with at least 1 question. The number of questions is returned with the group. The user is shown check boxes. When the select them, the UI is updated with categories to choose from the selected groups.</p>
<pre> <script> SELECT g.groupName, g.groupId, c.categoryName, c.categoryId, COUNT(*) AS QuestionCount FROM Question q JOIN Question_Category qc ON (q.questionId = qc.questionId) JOIN Category c ON (qc.categoryId = c.categoryId) JOIN `Group` g ON (c.groupId = g.groupId) WHERE 1 = 1 <if test='groupIds != null'> AND g.groupId IN <foreach item='groupId' index='groupIds' collection='groupIds' open='(' separator=',' close=')'> #{groupId} </foreach> </if> GROUP BY g.groupName, g.groupId, c.categoryName, c.categoryId HAVING COUNT(*) > 0 ORDER BY QuestionCount DESC, c.categoryName </script> </pre>	<p>This query is used to find a list of categories along with the number of questions in each category, where the number of questions is 1 or greater. This allows the user to see which categories they want to sample from when creating a game.</p> <p>Note: This query uses the extended MyBatis query format with script pre-processing.</p>
<pre> <script> SELECT q.* FROM Question q WHERE q.questionId IN (SELECT DISTINCT qc.questionId FROM Question_Category qc WHERE 1 = 1 <if test='categoryIds != null'> AND qc.categoryId IN <foreach item='categoryId' index='categoryIds' </pre>	<p>This query is used during game creation to randomly select questions to be used in a game from a specified set of categories. The MySQL rand() function is used for ordering and LIMIT is used to choose a limited number of items after the results are randomly ordered.</p>

<pre>collection='categoryIds' open='(' separator=',' close=')'> #{categoryId} </foreach> </if>) <choose> <when test='randomize'> ORDER BY rand() </when> <otherwise> ORDER BY q.questionId </otherwise> </choose> LIMIT #{maxResults} OFFSET #{offset} </script></pre>	
<pre>INSERT INTO Game (Created, EndTime, MaxQuestionTime, UserId) VALUES ({0}, {1}, {2}, {3})</pre>	Used to create a new game. A new unique id is assigned.
<pre>SELECT LAST_INSERT_ID()</pre>	Used to retrieve the newly created game id.
<pre>INSERT INTO Game_Question(GameQNum, GameId, QuestionId) VALUES ({0}, {1}, {2})</pre>	Adds one of the randomly sampled questions to the game.
<pre>INSERT INTO Game_Question_Answer(GameQANum, GameQNum, GameId, QuestionId, AnswerNumber) VALUES ({0}, {1}, {2}, {3}, {4})</pre>	Adds one of the randomly sampled possible answers choices to the game. This makes sure that every player of the same game sees the same set of possible answers and in the same order.

SQL Statements: Game Play

SQL Statements: Game Play - Find Games

In order to play a game, a user is shown a list of games that they can play. The following screenshot shows the game find page. Each game box may be selected to play a game. In each game box some information is show about the games.



SQL Statement	Purpose
<pre><script> SELECT GameExtended.* FROM (SELECT g.*, (SELECT COUNT(*) FROM Game_Question gq WHERE</pre>	<p>The query for finding games was rather complex. Much of this complexity came from wanting to give users information they would want about the game such as the number of people who played. It also has the ability to filter out games which are inactive or have already been completed by the user while still showing games in progress.</p>

```

        gq.GameId = g.GameId
    ) AS QuestionCount,
    (
        SELECT COUNT(*)
        FROM User_Games ug
        WHERE
            ug.GameId = g.GameId
    ) AS PlayerCount,
    ug_stats.timeStarted,
    COALESCE(ug_stats.PickCount, 0) AS PickCount,
    COALESCE(ug_stats.CorrectCount, 0) AS CorrectCount,
    COALESCE(ug_stats.InCorrectCount, 0) AS
IncorrectCount
FROM Game g
LEFT JOIN (
    SELECT
        ug.gameId,
        ug.timeStarted,
        COUNT(*) AS PickCount,
        SUM(CASE WHEN a.correct THEN 1 ELSE 0 END) AS
CorrectCount,
        SUM(CASE WHEN a.correct THEN 0 ELSE 1 END) AS
IncorrectCount
    FROM User_Games ug
    JOIN User_Game_QA_Pick p ON (ug.gameId =
p.gameId AND ug.userId = p.userId)
    JOIN Game_Question_Answer gqa ON (gqa.gameId =
p.gameId AND gqa.gameQANum = p.gameQANum AND
gqa.gameQNum = p.gameQNum)
    JOIN Answer a ON (a.questionId = gqa.questionId
AND a.answerNumber = gqa.answerNumber)
    WHERE
        <choose>
            <when test='userId != null'>
                ug.userId = #{userId}
            </when>
            <otherwise>
                ug.userId IS NULL
            </otherwise>
        </choose>
    GROUP BY ug.gameId, ug.timeStarted
) ug_stats ON (g.gameId = ug_stats.gameId)
) GameExtended
WHERE 1 = 1
<if test='request.filterInactive'>
    AND GameExtended.EndTime >
CURRENT_TIMESTAMP

```

This query was also reused in other parts of the application for showing games that were played, which include stats about the users performance for the games.

<pre> </if> <if test='userId != null && request.filterPlayed'> AND GameExtended.PickCount < GameExtended.QuestionCount </if> <if test='userId != null && request.filterUnPlayed'> AND GameExtended.PickCount > 0 </if> <choose> <when test='request.orderByTimeStarted'> ORDER BY GameExtended.timeStarted DESC </when> <otherwise> ORDER BY GameExtended.created DESC </otherwise> </choose> LIMIT #{request.maxResults} OFFSET #{request.offset} </script> </pre>	
<pre> <script> SELECT DISTINCT gq.gameId, c.categoryName FROM Game_Question gq JOIN Question_Category qc ON (gq.questionId = qc.questionId) JOIN Category c ON (qc.categoryId = c.categoryId) WHERE <choose> <when test='gameIds.size() > 0'> gq.GameId IN <foreach item='gameId' index='gameIds' collection='gameIds' open='(' separator=',' close=')'> #{gameId} </foreach> </when> <otherwise> 1 = 0 </otherwise> </choose> </script> </pre>	<p>This query was used to find out which categories are part of a game. This query was built to take a list of game ids and is executed right after the query above.</p> <p>One of the difficulties was that from a UI perspective this information should be returned back with the list of games and for performance reasons it was helpful to issue one query for all game ids at once.</p>

SQL Statements: Game Play - Play Game

The following UI screenshots show important aspects of game play for the application. The SQL statements below them were used to implement the user interface.

Showing a question during game play.



The screenshot shows a web application interface for 'Rocket Trivia'. At the top, there is a dark grey header with a rocket icon and the text 'Rocket Trivia'. Below the header, the main content area has a light grey background. The title 'Answer Question' is displayed in a large, bold, black font. Below the title, a question is presented in an italicized black font: 'In Terraria, which of the following items does the Martian Saucer mini-boss NOT drop?'. There are four rounded rectangular buttons stacked vertically, each containing one of the following items: 'Cosmic Car Key', 'Anti-Gravity Hook', 'Influx Waver', and 'Drill Containment Unit'. At the bottom of the page, there is a footer with the links 'Home | Sign Out | Profile' in a blue font.

Showing the answer screen.




SQL Statement	Purpose
<pre>SELECT ug.* FROM User_Games ug WHERE ug.userId = #{0} AND ug.gameId = #{1}</pre>	When game play begins the application uses this query to see if the user has already begun playing the game.
<pre>INSERT INTO User_Games (userId, gameId, timeStarted) VALUES (#{0}, #{1}, CURRENT_TIMESTAMP)</pre>	If the user has not yet joined a game, this query will insert a record showing the user joined a game.
<pre>SELECT p.*, a.correct, a.content FROM User_Game_QA_Pick p JOIN Game_Question_Answer gqa ON (gqa.gameId = p.gameId AND gqa.gameQNum = p.gameQNum AND gqa.gameQANum = p.gameQANum) JOIN Answer a ON (gqa.questionId = a.questionId AND gqa.answerNumber = a.answerNumber) WHERE p.userId = #{0} AND p.gameId = #{1}</pre>	If the user has started the game, the previous choices are found with the following query. The results allow the UI to know which question to go to next and how to show stats about how the user is current doing while playing the game.
<pre>SELECT gq.*, q.content FROM Game_Question gq JOIN Question q ON (gq.questionId = q.questionId) WHERE gq.gameId = #{0} ORDER BY gq.GameQNum</pre>	The following query is used to retrieve the questions for the game. During game play these results are compared against previous picks to know which question to present next. The questions are also presented in the order given by this query.

<pre>SELECT gqa.*, a.content, a.correct FROM Game_Question_Answer gqa JOIN Answer a ON (gqa.questionId = a.questionId AND gqa.answerNumber = a.answerNumber) WHERE gqa.gameId = #{0} AND gqa.gameQNum = #{1} ORDER BY gqa.gameQANum</pre>	<p>During game play when each question is shown, the answer choices are retrieved using this query. This includes which order they should be shown in.</p>
<pre>INSERT INTO User_Game_QA_Pick (userId, gameId, gameQNum, gameQANum, timeChosen, timeElapsed) VALUES (#{pick.userId}, #{pick.gameId}, #{pick.gameQNum}, #{pick.gameQANum}, #{pick.timeChosen}, #{pick.timeElapsed})</pre>	<p>The following insert is used when the user chooses an answer. This records the users answer choice for a question in a game.</p>

SQL Statements: Game Play - Game Review

Once a user has played a game, they can review their game performance, see what the right answers were, and compare it to others who played the same game.

Showing Game Review

 **Rocket Trivia**

Review Game

Question 1

Which letter do you need to have on a European driver license in order to ride any motorbikes?

Your answer was:

D

The correct answer was:

A

It took you 8 seconds to answer the question.
You answered this question on 3/11/2018 at 7:13 pm.

Question 2


Which of the following Pacific Islander countries is ruled by a constitutional monarchy?

Your answer was:

Tonga

It took you 4 seconds to answer the question.
You answered this question on 3/11/2018 at 7:13 pm.

Showing a Leader Board for a single game.



Rocket Trivia

Game Player Stats

Rank	Alias	Percent Correct	Questions Answered
1	bryan	100.0	5
1	bob	100.0	5
3	';alert(123);t='	60.0	5
4	mrcrane	40.0	5

[Home](#) | [Sign Out](#) | [Profile](#)

SQL Statement	Purpose
<pre> SELECT p.gameId, gq.gameQNum AS QNum, q.content AS Question, a.Content AS ChosenAnswer, aRight.Content AS CorrectAnswer, a.Correct, p.TimeChosen, p.TimeElapsed FROM User_Game_QA_Pick p JOIN Game_Question gq ON (gq.gameId = p.gameId AND gq.gameQNum = p.gameQNum) JOIN Question q ON (gq.questionId = q.questionId) JOIN Game_Question_Answer gqa ON (gqa.gameId = p.gameId AND gqa.gameQNum = p.gameQNum AND gqa.GameQANum = p.GameQANum) JOIN Answer a ON (a.QuestionId = gqa.QuestionId AND a.AnswerNumber = gqa.AnswerNumber) JOIN (Game_Question_Answer gqaRight JOIN Answer aRight ON (gqaRight.QuestionId = aRight.QuestionId AND gqaRight.AnswerNumber = aRight.AnswerNumber AND aRight.Correct = 1) </pre>	<p>This query returns all of the answered questions for a particular user in a particular game. It also shows for each question if the user was correct or not. It also shows what the correct answer was.</p> <p>This query is used to review a game that the user played.</p>

<pre>) ON (gqaRight.GameId = p.gameId AND gqaRight.GameQNum = p.GameQNum) WHERE p.userId = #{0} AND p.gameId = #{1} ORDER BY gq.gameQNum </pre>	
<pre> SELECT UG_Stats.gameId, UG_Stats.userId, UG_Stats.alias, 100 * UG_Stats.NumCorrect / G_Stats.QuestionCount AS PercentCorrect, UG_Stats.NumCorrect, UG_Stats.NumAnswered, G_Stats.QuestionCount FROM (SELECT ug.gameId, ug.userId, u.alias, SUM(a.correct) AS NumCorrect, COUNT(*) AS NumAnswered FROM User_Games ug JOIN User u ON (ug.userId = u.userId) JOIN User_Game_QA_Pick p ON (p.userId = ug.userId AND p.gameId = ug.gameId) JOIN Game_Question_Answer gqa ON (gqa.gameId = p.gameId AND gqa.gameQNum = p.gameQNum AND gqa.gameQANum = p.gameQANum) JOIN Answer a ON (a.questionId = gqa.questionId AND a.answerNumber = gqa.answerNumber) WHERE ug.gameId = #{gameId} GROUP BY ug.gameId, ug.userId, u.alias) UG_Stats JOIN (SELECT g.gameId, </pre>	<p>This query shows a leader board for a game. It shows from top to bottom the users who did the best in a game.</p> <p>A user sees these results to see how well they did against other people.</p> <p>One difficulty we ran into with producing rankings was that MySQL did not have a function to assign a rank on query output. We had to post process the query results in Java to assign ranks.</p>

<pre>COUNT(*) AS QuestionCount FROM Game g JOIN Game_Question gq ON (gq.gameId = g.gameId) WHERE g.gameId = #{gameId} GROUP BY g.gameId) G_Stats ON (UG_Stats.gameId = G_Stats.gameId) ORDER BY UG_Stats.NumCorrect DESC, UG_Stats.NumAnswered DESC LIMIT #{topNumber}</pre>	
---	--

SQL Statements: User Statistics

The application shows some very basic statistics for a user on the profile page. It shows statistics such as the percentage of questions they answered right for each category. This lets the user see where they do well and where they need some work.

Question Category Performance

Rank	Category	Percent Correct	# Correct	# Incorrect
1	Vehicles	100.0	1	0
2	Science	100.0	1	0
3	Mythology	92.3	12	1
4	General Knowledge	57.1	4	3
5	Science: Mathematics	52.6	10	9
6	Politics	50.0	6	6
7	Animals	50.0	4	4
8	Entertainment: Comics	50.0	3	3
9	Science & Nature	50.0	2	2
10	Science: Computers	37.5	3	5
11	Entertainment: Film	36.4	4	7
12	Entertainment: Television	35.0	7	13
13	Science: Gadgets	33.3	1	2
14	Entertainment: Cartoon & Animations	25.0	1	3
15	Entertainment: Video Games	18.2	8	36
16	Movies	0.0	0	1
17	Math	0.0	0	1

SQL Statement	Purpose
<pre>SELECT CategoryStats.* FROM (SELECT c.CategoryName, SUM(QStats.CorrectCount) AS CorrectCount, SUM(QStats.IncorrectCount) AS IncorrectCount, SUM(QStats.TimesAnswered) AS AnswerCount, 100 * SUM(QStats.CorrectCount) / SUM(QStats.TimesAnswered) AS PercentCorrect FROM (SELECT a.QuestionId, SUM(a.Correct) AS CorrectCount, (COUNT(*) - SUM(a.Correct)) AS IncorrectCount,</pre>	<p>This query is used to show the screenshot above. It shows how well a user does in certain trivia categories.</p>

<pre> COUNT(*) AS TimesAnswered FROM User_Games ug JOIN User_Game_QA_Pick p ON (p.gameld = ug.gameld AND p.userId = ug.userId) JOIN Game_Question_Answer gqa ON (gqa.Gameld = p.Gameld AND gqa.GameQNum = p.GameQNum AND gqa.GameQANum = p.GameQANum) JOIN Answer a ON (a.QuestionId = gqa.QuestionId AND a.AnswerNumber = gqa.AnswerNumber) WHERE ug.userId = #{userId} GROUP BY a.QuestionId) QStats JOIN Question_Category qc ON (qc.QuestionId = QStats.QuestionId) JOIN Category c ON (c.categoryId = qc.categoryId) GROUP BY c.CategoryName) CategoryStats ORDER BY CategoryStats.PercentCorrect DESC, CategoryStats.CorrectCount DESC </pre>	
--	--

SQL Statements: Global Leaderboard

The application has a very basic global ranking system based on points. The application awards users with points for questions answered correctly. The speed of answering a question also matters. The faster they answer the more points they receive. Questions answered incorrectly will remove points for the user.

The global leaderboard shows the top 20 users for the site by their point ranking.

Trivia Leader Board

Rank	Alias	Points	Games	Questions	% Correct	Avg. Response Time
1	bryan	172.5	14	163	41	4
2	Wizbiz	67.0	7	61	39	4
3	arisomers	50.0	2	20	70	6
4	Kaylie	42.6	3	70	37	10
5	';alert(123);t='	36.0	2	15	67	5
6	mrcrane	32.0	3	25	40	8
7	cc	31.0	2	20	55	4
8	bob	21.0	2	6	83	129
9	TechFatih	20.0	2	20	45	6
10	nibblar	16.5	2	20	70	18
11	JamesC	16.0	1	10	50	6
12	Banana Mike	15.5	1	10	60	9
13	Trece	12.5	2	15	47	12
14	CSS451Tester	8.0	1	10	30	1
15	E	-1.0	1	1	0	3
16	artursouza	-2.0	2	20	25	9
17	Kmc	-3.0	1	3	0	12

SQL Statement	Purpose
<pre>SELECT p.UserId, u.Alias, SUM(CASE WHEN NOT a.correct THEN -1 WHEN p.TimeElapsed <= 5 THEN 5 WHEN p.TimeElapsed <= 10 THEN 2 WHEN p.TimeElapsed <= 30 THEN 1 WHEN p.TimeElapsed <= 60 THEN 0.5 ELSE 0.1 END) AS Points, COUNT(*) AS NumAnswered, SUM(a.correct) AS NumCorrect, COUNT(*) - SUM(a.correct) AS NumIncorrect, COUNT(DISTINCT p.gameId) AS GamesPlayed,</pre>	<p>This query implements the global leaderboard. It computed the points for questions and sums up points by user. The results are ordered by points.</p> <p>As in other queries a rank number could not be assigned using MySQL. The results were post processed to add a rank number. The java code made sure that users with the same point number had the same rank. You may see 2 users with rank #1 and the next user down would have rank #3.</p>

```

AVG(p.TimeElapsed) AS AverageAnswerTime
FROM User_Game_QA_Pick p
JOIN User u ON (
  u.userId = p.userId)
JOIN Game_Question_Answer gqa ON (
  gqa.gameId = p.gameId AND
  gqa.gameQNum = p.gameQNum AND
  gqa.gameQANum = p.gameQANum)
JOIN Answer a ON (
  a.questionId = gqa.questionId AND
  a.answerNumber = gqa.answerNumber)
GROUP BY p.UserId, u.Alias
ORDER BY Points DESC
LIMIT 20

```

Normalization

USER

FD1: UserID \rightarrow {Email, Alias, PasswordHash}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

GROUP

FD1: GroupID \rightarrow {GroupName, GroupDescription}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.

- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

CATEGORY

FD1: CategoryID \rightarrow {CategoryName, GroupID}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

QUESTION

FD1: QuestionID \rightarrow {Content, GroupID}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

ANSWER

FD1: {AnswerNumber, QuID} \rightarrow {Correct, Content}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

GAME

FD1: GameID \rightarrow {Created, EndTime, MaxQuestionTime}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

GAME_QUESTION

FD1: {GamesQA, GameID} \rightarrow QuestionID

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

GAME_QUESTION_ANSWER

FD1: {GameQANumber, GameQNum, GameID} \rightarrow {QuestionID, AnswerNumber}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

USER_GAMES

FD1: {UserID, GameID} \rightarrow TimeStarted

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

GROUP_ADMINISTRATORS

FD: No Dependencies...

NF: NF1

The reasons that this relation is NF1 are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Doesn't Satisfy NF2, NF3, or BCNF - There are no dependencies.

This is because this specific relation is represented using two attributes, both of which create the primary key. We wouldn't want to further normalize it because this would result in a loss of vital data used within our web application.

USER_GAME_QA_PICK

FD1: {UserID, GameID, GameQNum, GameQANum} \rightarrow {TimeChosen, TimeElapsed}

NF: BCNF

The reasons that this relation is BCNF are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Satisfies NF2 Requirements - Every non-primary-key attribute is fully dependent on the primary key.
- Satisfies NF3 Requirements - It is NF2 and all non-primary-key attribute is directly dependent on the primary key.
- Satisfies BCNF - It is both NF2 and NF3. More importantly there is no functional dependency where $X \rightarrow A$ and A is a prime attribute of R.

QUESTION_CATEGORIES

FD: No Dependencies...

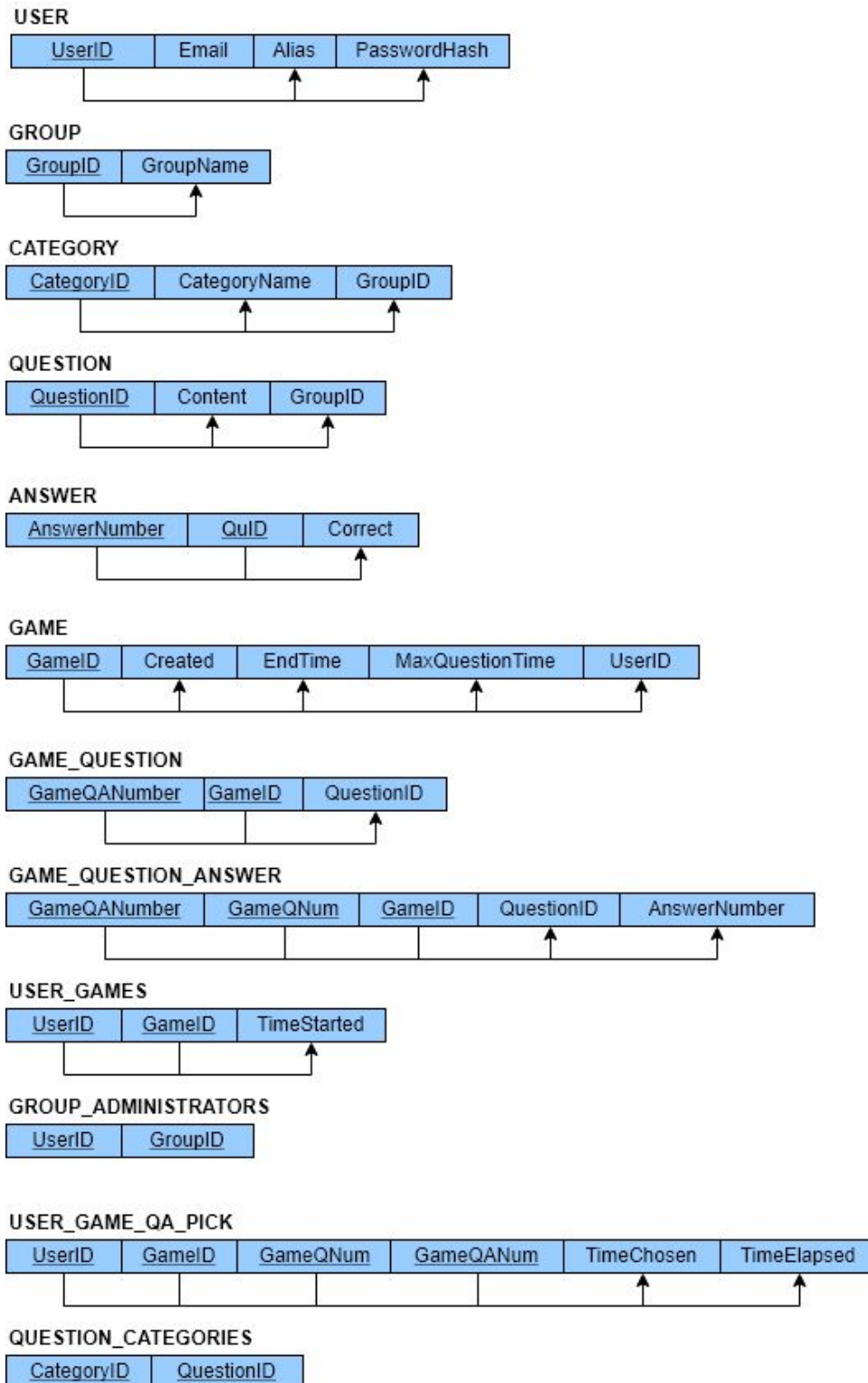
NF: NF1

The reasons that this relation is NF1 are due to the following:

- Satisfies NF1 Requirements - There are no multivalued or nested relations.
- Doesn't Satisfy NF2, NF3, or BCNF - There are no dependencies.

This is because this specific relation is represented using two attributes, both of which create the primary key. We wouldn't want to further normalize it because this would result in a loss of vital data used within our web application.

Normalized Schema



Project Evaluation and Reflection

The overall project went smoothly from a design perspective. The team put in early effort into the ER diagrams and the relational model. As a team, we designed the tables and reviewed our diagrams. This upfront work paid off considerably and we only had to make minor adjustments to the attributes in the tables. No changes were needed for the relationships themselves.

One of the early discussions we had during the design process was about weak entities. For categories and questions we used strong entities even though they could have been weak entities under Group. With game questions and game answers we did use weak entities. We found that our queries involving strong entities which much simpler and we needed less complex joins. This is a topic we would reconsider when building another application.

Another aspect of normalization we found was that some queries were very difficult with our schema. Assessing the performance of a game was difficult because the correctness of an answer was not recorded with a users choice. The user choose an answer and the correctness was part of the answer and not the choice. It would have been fairly simple to denormalize a few attributes and have them repeated in other tables. This would have made many queries simpler and possibly faster.

One of the other pain points the team ran into was use of MySQL. While it was easy to set up for local testing and for deployment on AWS there were some issues. One of our biggest issues was related to check constraints. We wrote check constraints for the database which MySQL accepted with no error. Later on we found out they did not work. We did not test our constraints well. We did research and found out that would could use triggers instead. One other issue we ran into with MySQL was support for ranking. We had several queries in which a rank could be assigned. We could not find a way to assign a rank number using SQL. We had to post process results in Java to add ranks. We noticed that MySQL seemed to lack features of other databases. We did some basic research and saw that PostgreSQL had some of these missing features. PostgreSQL is also open source, easy to setup, and is supported on AWS and Azure. It is also well supported using the application tools we used to build the application, java, Hikari CP, and MyBatis. If we were to build a new application we would consider PostgreSQL before MySQL.

Overall the project was rather time consuming. The database design and schema generation was one of the simpler aspects of the project. The course homework and class samples helped us make a proper design and translating to SQL table creation statements was easy. Much more work went into building the application. A large amount of time was spent setting up AWS. While MySQL setup was easy, we had to create VMs for the application, setup a load balancer, create a domain name, and obtain an SSL certificate.

The other place where time was spent was the development of the user interface. We choose a UI framework that no one on the team had used, Vue.js. One of the team members, bryan, was able to pick it up easily due to its similarity with other UI frameworks. None of the other developers were familiar

enough with web development though to pick up the technology. Even though it was difficult, we would probably have not changed our decision since any UI framework would have been difficult for the team to pick up.

As a team, managing our work was difficult. We did not stick to the implementation plan very well. We did not do our work in the order we thought and we did not split up work as we thought. Even so, we were able to complete the application and it worked well enough that we felt comfortable asking other users on facebook and from the class to try the application out.

Implementation Plan

Schedule

The following schedule with milestones will be used to complete the project. Our first priority is to figure out the application requirements and bootstrap the development environment so that multiple individuals can begin work in parallel. During the second phase we will start implementing the user interface in iterations while figuring out the database design. This will help us make sure that the database design and use cases go together well. We will then build out the database design and hook up REST APIs corresponding to the use cases identified during UI design. We will target completion of the application 1 week prior to the end date. We would like to leave some time buffer for documentation and fixing bugs.

Date	Goal
1/12/2018	Proposal for software tooling.
1/15/2018	Skeleton for application and instructions for local database installation created.
1/15/2018	Gitlab group and source repository completed.
1/22/2018	Finalize requirements
1/25/2018	UI identity handling with mock user data.
1/26/2018	ER Model and Relation Model
2/1/2018	Final choice on all development tools
2/7/2018	UI mockups created
2/15/2018	Development tools write-up

2/18/2018	Prototype UI completed
2/22/2018	Create final database schema
2/26/2018	Sample data including load scripts
2/28/2018	Database created on the Cloud (AWS) with initial sample data.
3/1/2018	SQL query statements created including binding to java classes.
3/3/2018	REST API Complete
3/5/2018	Completed Development with Few Bugs
3/12/2018	Present Final Group Poster Session

Areas of Work

The team will assign logical areas of work to certain individuals. Individuals are responsible for generating ideas, driving discussion, and implementation in the specified areas. Every member in the group will be expected to review and discuss work in a given area, but it is the job of the owner to drive communication, the decision making process, and the final deliverable in a given area.

Not all areas of work will have an owner at the beginning of the project. Each week the team will review the areas of work and evaluate ownership. Owners may choose to swap areas.

The areas of work including REST APIs also includes the creation of SQL queries and binding those queries to the REST APIs.

Name	Description	Owners
Development Environment	The overall tools and technologies used for the application.	Bryan Castillo
Cloud Hosting	Setting up the production environment and deployment.	Tréce Wicklander-Bryant
Functional Requirements	Documenting requirements specification.	Fahad Alshehri
ER Model (Conceptual)	Responsible for the conceptual database design.	Tréce Wicklander-Bryant
Schema (Physical)	Responsible for the actual database schema.	Fahad Alshehri

User Interface Mockups	Creating basic UI mockups and UI flows.	Fatih Ridha
User Authentication	Responsible for handling user authentication from a browser perspective and for providing utilities and the application level for authentication and authorization.	Bryan Castillo
UI: User Creation	User interface for creating users including any user edit features.	Bryan Castillo
UI: Group Creation	UI for creating and modifying trivia groups.	Tréce Wicklander-Bryant
UI: Question Creation	UI for creating new questions including association to categories.	Fatih Ridha
UI: Category Creation	UI for creating and editing categories.	Fahad Alshehri
User Creation and Lookup REST APIs	Responsible for creating REST APIs to manage users.	Fahad Alshehri
Group REST APIs	Responsible for creating REST APIs to manage trivia groups.	Tréce Wicklander-Bryant
Category REST APIs	Responsible for creating REST APIs to manage trivia groups.	Fahad Alshehri
Question REST APIs	Responsible for creating REST APIs to manage questions.	Fatih Ridha
UI: Game Creation	The user interface for creating a new game.	Fatih Ridha
Game Creation REST APIs	APIs needed to create or help create trivia games.	Fatih Ridha
Game Question Answer REST APIs	APIs for providing answers to game questions.	Fatih Ridha
UI: Game Execution	The main user interface for participants playing a game.	Bryan Castillo
UI: Game Statistics	The user interface for showing game statistics.	Tréce Wicklander-Bryant
Game Statistics API	The REST APIs for retrieving game statistics.	Tréce Wicklander-Bryant

UI: General Question Statistics	The user interface for showing information about questions and how well they are answered.	Fahad Alshehri
General Question REST APIs	The REST APIs for retrieving question statistics.	Tréce Wicklander-Bryant
Test Plan	Responsible for creating a manual end to end test play or user acceptance test.	Bryan Castillo
End to End testing	Responsible for running user acceptance tests and filing issues.	Bryan Castillo
Poster and demo organization.	Responsible for the creation of posters and running live demos.	Fahad Alshehri

Appendix

Schema Creation Statements

Schema and Users

The following SQL should be run to create the logical database schema, the users, and the permissions to the database users.

```
DROP SCHEMA IF EXISTS rocket;
```

```
CREATE SCHEMA IF NOT EXISTS rocket;
```

```
CREATE USER 'rocket'@'%' IDENTIFIED BY 'kN78z-ng~148A';
CREATE USER 'rocket'@'localhost' IDENTIFIED BY 'kN78z-ng~148A';
GRANT ALL PRIVILEGES ON rocket.* TO 'rocket'@'%';
GRANT ALL PRIVILEGES ON rocket.* TO 'rocket'@'localhost';
GRANT ALTER ON rocket.* TO 'rocket'@'localhost';
GRANT ALTER ON rocket.* TO 'rocket'@'%';
```

```
CREATE USER 'css451-tester'@'%' IDENTIFIED BY 'cuniculus';
CREATE USER 'css451-tester'@'localhost' IDENTIFIED BY 'cuniculus';
GRANT ALL PRIVILEGES ON rocket.* TO 'css451-tester'@'%';
GRANT ALL PRIVILEGES ON rocket.* TO 'css451-tester'@'localhost';
GRANT ALTER ON rocket.* TO 'css451-tester'@'localhost';
GRANT ALTER ON rocket.* TO 'css451-tester'@'%';
```

Table Creation Statements

The following section provides the code that was used to create all the tables contained within our schema.

USER

```
CREATE TABLE User (
  UserId INTEGER NOT NULL AUTO_INCREMENT,
  Email VARCHAR(200) NOT NULL,
  Alias VARCHAR(100) NOT NULL,
  PasswordHash VARCHAR(100) NOT NULL,
  PRIMARY KEY(UserId),
  UNIQUE(Email)
);
```

GROUP

```
CREATE TABLE `Group` (
  GroupId INTEGER NOT NULL AUTO_INCREMENT,
  GroupName VARCHAR(100) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(GroupId),
  UNIQUE(GroupName)
);
```

GROUP_ADMINISTRATOR

```
CREATE TABLE Group_Administrator (
  UserId INTEGER NOT NULL,
  GroupId INTEGER NOT NULL,
  PRIMARY KEY(UserId, GroupId),
  FOREIGN KEY (UserId) REFERENCES User (UserId),
  FOREIGN KEY (GroupId) REFERENCES `Group` (GroupId)
);
```

QUESTION

```
CREATE TABLE Question (
  QuestionId INTEGER NOT NULL AUTO_INCREMENT,
  Content VARCHAR(500) NOT NULL,
  GroupId INTEGER NOT NULL,
  PRIMARY KEY(QuestionId),
  FOREIGN KEY (GroupId) REFERENCES `Group` (GroupId)
);
```

ANSWER

```
CREATE TABLE Answer (
  AnswerNumber INTEGER NOT NULL,
  QuestionId INTEGER,
  Content VARCHAR(500) NOT NULL,
  Correct BOOLEAN NOT NULL,
  PRIMARY KEY(AnswerNumber, QuestionId),
  FOREIGN KEY(QuestionId) REFERENCES Question (QuestionId)
);
```

CATEGORY

```
CREATE TABLE Category (
  CategoryId INTEGER NOT NULL AUTO_INCREMENT,
  CategoryName VARCHAR(100) NOT NULL,
  GroupId INTEGER NOT NULL,
  PRIMARY KEY(CategoryId),
  UNIQUE(CategoryName, GroupId),
  FOREIGN KEY (GroupId) REFERENCES `Group` (GroupId)
);
```

QUESTION_CATEGORY

```
CREATE TABLE Question_Category (
  CategoryId INTEGER NOT NULL,
  QuestionId INTEGER NOT NULL,
  PRIMARY KEY(CategoryId, QuestionId),
  FOREIGN KEY(CategoryId) REFERENCES Category (CategoryId),
  FOREIGN KEY(QuestionId) REFERENCES Question (QuestionId),
  CONSTRAINT CHECK_SAME_GROUP
  CHECK (
    (SELECT c.groupId FROM Category c WHERE c.categoryId = categoryId) =
    (SELECT q.groupId FROM Question q WHERE q.categoryId = categoryId)
  )
);
```

GAME

```
CREATE TABLE Game (
  GameId INTEGER NOT NULL AUTO_INCREMENT,
  Created TIMESTAMP NOT NULL,
  EndTime TIMESTAMP NOT NULL,
  MaxQuestionTime INT NOT NULL,
  UserId INT NOT NULL,
  CONSTRAINT GamePK
    PRIMARY KEY(GameId),
  CONSTRAINT GameFK
    FOREIGN KEY(UserId) REFERENCES User (UserId),
  CONSTRAINT GameValidEndTime
    CHECK (EndTime > Created),
  CONSTRAINT GameValidMaxTime
    CHECK (MaxQuestionTime > 0)
);
```

GAME_QUESTION

```
CREATE TABLE Game_Question (
  GameQNum INT NOT NULL,
  GameId INT NOT NULL,
  QuestionId INT NOT NULL,
  CONSTRAINT GameQuestionPK
    PRIMARY KEY(GameQNum, GameId),
  CONSTRAINT GameQuestionFKGID
    FOREIGN KEY(GameId) REFERENCES Game (GameId),
  CONSTRAINT GameQuestionFKQID
    FOREIGN KEY(QuestionId) REFERENCES Question (QuestionId)
);
```

GAME_QUESTION_ANSWER

```
CREATE TABLE Game_Question_Answer (
  GameQANum INT NOT NULL,
  GameQNum INT NOT NULL,
  GameId INT NOT NULL,
  QuestionId INT NOT NULL,
```

```

AnswerNumber INT NOT NULL,
CONSTRAINT GameQuestionAnswerPK
    PRIMARY KEY(GameQANum, GameQNum, GameId),
CONSTRAINT GameQuestionAnswerFKGQ
    FOREIGN KEY(GameQNum, GameId) REFERENCES Game_Question (GameQNum, GameId),
CONSTRAINT GameQuestionAnswerFKAnswer
    FOREIGN KEY(AnswerNumber, QuestionId) REFERENCES Answer (AnswerNumber, QuestionId)
);

```

USER_GAMES

```

CREATE TABLE User_Games (
    UserId INT NOT NULL,
    GameId INT NOT NULL,
    TimeStarted TIMESTAMP NOT NULL,
    CONSTRAINT UserGamePK
        PRIMARY KEY(UserId, GameId),
    CONSTRAINT UserGameFKUser
        FOREIGN KEY(UserId) REFERENCES User (UserId),
    CONSTRAINT UserGameFKGame
        FOREIGN KEY(GameId) REFERENCES Game (GameId)
);

```

USER_GAME_QA_PICK

```

CREATE TABLE User_Game_QA_Pick (
    UserId INT NOT NULL,
    GameId INT NOT NULL,
    GameQNum INT NOT NULL,
    GameQANum INT NOT NULL,
    TimeChosen TIMESTAMP NOT NULL,
    TimeElapsed INT NOT NULL,
    CONSTRAINT UserGameQAPickPK
        PRIMARY KEY(UserId, GameId, GameQNum, GameQANum),
    CONSTRAINT UserGameQAPickFKUID
        FOREIGN KEY(UserId) REFERENCES User (UserId),
    CONSTRAINT UserGameQAPickFKGQA
        FOREIGN KEY(GameQANum, GameQNum, GameId) REFERENCES Game_Question_Answer (GameQANum, GameQNum, GameId)
);

```

Trigger Creation Statements

The following section provides the code that was used to create all the triggers for each table.

USER

```

-- TRIGGERS FOR INSERTING AN EMAIL, ALIAS, AND PASSWORD HASH
delimiter $
create trigger USER_CHECK_INS before insert on User
for each row
begin
    if (CHAR_LENGTH(new.Email) < 1 OR new.Email REGEXP '^[[:space:]]' OR new.Email REGEXP '[[:space:]]$')
    then

```

```

        signal sqlstate '45000' set message_text = 'Email Error';

        elseif (CHAR_LENGTH(new.Alias) < 1 OR new.Alias REGEXP '^[:space:]' OR new.Alias REGEXP
'[:space:]]$')
        then
            signal sqlstate '45000' set message_text = 'Alias Error';

        elseif (CHAR_LENGTH(new.PasswordHash) < 1 OR new.PasswordHash REGEXP '^[:space:]' OR
new.PasswordHash REGEXP '[:space:]]$')
        then
            signal sqlstate '45000' set message_text = 'Password Error';

        end if;
    end$

-- TRIGGERS FOR UPDATING AN EMAIL, ALIAS, AND PASSWORD HASH
create trigger USER_CHECK_UP before update on User
for each row
begin
    if (CHAR_LENGTH(new.Email) < 1 OR new.Email REGEXP '^[:space:]' OR new.Email REGEXP '[:space:]]$')
    then
        signal sqlstate '45000' set message_text = 'Email Error';

    elseif (CHAR_LENGTH(new.Alias) < 1 OR new.Alias REGEXP '^[:space:]' OR new.Alias REGEXP
'[:space:]]$')
    then
        signal sqlstate '45000' set message_text = 'Alias Error';

    elseif (CHAR_LENGTH(new.PasswordHash) < 1 OR new.PasswordHash REGEXP '^[:space:]' OR
new.PasswordHash REGEXP '[:space:]]$')
    then
        signal sqlstate '45000' set message_text = 'Password Error';

    end if;
end$

delimiter ;

```

GROUP

```

-- TRIGGERS FOR INSERTING A GroupName and Group Description
delimiter $
create trigger GROUP_TRIGGER_INS before insert on `Group`
for each row
begin
    if (CHAR_LENGTH(new.GroupName) < 1 OR new.GroupName REGEXP '^[:space:]' OR new.GroupName REGEXP
'[:space:]]$')
    then
        signal sqlstate '45000' set message_text = 'GroupName Error';
    end if;
end$

```

```

        elseif (CHAR_LENGTH(new.Description) < 1 or new.Description REGEXP '^[:space:]' OR new.Description
REGEXP '[:space:]$')
        then
            signal sqlstate '45000' set message_text = 'Description Error';
        end if;
    end$

-- TRIGGERS FOR UPDATING A GroupName and Group Description
delimiter $
create trigger GROUP_TRIGGER_UP before update on `Group`
for each row
begin
    if (CHAR_LENGTH(new.GroupName) < 1 OR new.GroupName REGEXP '^[:space:]' OR new.GroupName REGEXP
'[:space:]$')
    then
        signal sqlstate '45000' set message_text = 'Password Error';

        elseif (CHAR_LENGTH(new.Description) < 1 or new.Description REGEXP '^[:space:]' OR new.Description
REGEXP '[:space:]$')
        then
            signal sqlstate '45000' set message_text = 'Password Error';
        end if;
    end$

delimiter ;

```

QUESTION

```

-- Triggers for Question
delimiter $
create trigger QUESTION_TRIGGER_INS before insert on Question
for each row
begin
    if (CHAR_LENGTH(new.Content) < 1 OR new.Content REGEXP '^[:space:]' OR new.Content REGEXP
'[:space:]$')
    then
        signal sqlstate '45000' set message_text = 'Question Error';

        elseif (EXISTS(SELECT 1 FROM QUESTION WHERE GroupID = new.GroupID and Content = new.Content))
        then
            signal sqlstate '45000' set message_text = 'Question Error';
        end if;
    end$

delimiter $
create trigger QUESTION_TRIGGER_UP before update on Question
for each row
begin

```

```

    if (CHAR_LENGTH(new.Content) < 1 OR new.Content REGEXP '^[:space:]' OR new.Content REGEXP
'[:space:]$')
    then
        signal sqlstate '45000' set message_text = 'Question Error';

    elseif (EXISTS(SELECT 1 FROM QUESTION WHERE GroupID = new.GroupID and Content = new.Content))
    then
        signal sqlstate '45000' set message_text = 'Question Error';
    end if;
end$

```

ANSWER

```

-- Triggers for Answer
delimiter $
create trigger ANSWER_TRIGGER_INS before insert on Answer
for each row
begin
    if (CHAR_LENGTH(new.Content) < 1 OR new.Content REGEXP '^[:space:]' OR new.Content REGEXP
'[:space:]$') OR (AnswerNumber >= 0)
    then
        signal sqlstate '45000' set message_text = 'Answer Error';

    elseif (EXISTS(SELECT 1 FROM ANSWER WHERE QuestionId = new.QuestionId and Content = new.Content))
    then
        signal sqlstate '45000' set message_text = 'Answer Error';
    end if;
end$

delimiter $
create trigger ANSWER_TRIGGER_UP before update on Answer
for each row
begin
    if (CHAR_LENGTH(new.Content) < 1 OR new.Content REGEXP '^[:space:]' OR new.Content REGEXP
'[:space:]$') OR (AnswerNumber >= 0)
    then
        signal sqlstate '45000' set message_text = 'Answer Error';

    elseif (EXISTS(SELECT 1 FROM ANSWER WHERE QuestionId = new.QuestionId and Content = new.Content))
    then
        signal sqlstate '45000' set message_text = 'Answer Error';
    end if;
end$

delimiter ;

```

CATEGORY

```

-- Triggers for Category
delimiter $

```

```

create trigger CATEGORY_TRIGGER_INS before insert on Category
for each row
begin
    if (CHAR_LENGTH(new.CategoryName) < 1 OR new.CategoryName REGEXP '^[:space:]' OR new.CategoryName
REGEXP '[:space:]$')
    then
        signal sqlstate '45000' set message_text = 'Category Error';
    end if;
end$

delimiter $
create trigger CATEGORY_TRIGGER_UP before update on Category
for each row
begin
    if (CHAR_LENGTH(new.CategoryName) < 1 OR new.CategoryName REGEXP '^[:space:]' OR new.CategoryName
REGEXP '[:space:]$')
    then
        signal sqlstate '45000' set message_text = 'Category Error';
    end if;
end$

```

USER_GAMES

-- Triggers for User Game

```

delimiter $
create trigger USER_GAME_TRIGGER_INS before insert on USER_GAMES
for each row
begin
    if NOT (new.TimeStarted BETWEEN
        (SELECT Created FROM Game G WHERE G.GameId = GameID) AND
        (SELECT EndTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game Error';
    end if;
end$

delimiter $
create trigger USER_GAME_TRIGGER_UP before update on USER_GAMES
for each row
begin
    if NOT (new.TimeStarted BETWEEN
        (SELECT Created FROM Game G WHERE G.GameId = GameID) AND
        (SELECT EndTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game Error';
    end if;
end$

```


USER_GAME_QA_PICK

-- Triggers for User Game QA Pick

```

delimiter $
create trigger USER_GAME_QA_PICK_TRIGGER_INS before insert on USER_GAME_QA_PICK
for each row
begin
    if NOT (new.TimeChosen BETWEEN
        (SELECT Created FROM Game G WHERE G.GameId = GameID) AND
        (SELECT EndTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game QA Pick Error';

    elseif NOT (new.TimeElapsed BETWEEN 0 AND (SELECT MaxQuestionTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game QA Pick Error';
    end if;
end$

delimiter $
create trigger USER_GAME_QA_PICK_TRIGGER_UP before update on USER_GAME_QA_PICK
for each row
begin
    if NOT (new.TimeChosen BETWEEN
        (SELECT Created FROM Game G WHERE G.GameId = GameID) AND
        (SELECT EndTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game QA Pick Error';

    elseif NOT (new.TimeElapsed BETWEEN 0 AND (SELECT MaxQuestionTime FROM Game G WHERE G.GameId = GameID))
    then
        signal sqlstate '45000' set message_text = 'User Game QA Pick Error';
    end if;
end$
delimiter ;

```

Basic Initial Data Population

The first set of data populated in the test database setup some basic test users, groups, categories, and a few questions. This helped write enough of the application so that more content could be generated through the application itself.

The basic SQL inserts setup users which store a hash of the user's password. In order to compute the password for manual insertion the following python script, get-password-hash.py, can be used.

```

import sys
import hashlib

```

```
import base64

if len(sys.argv) != 2:
    print("Usage: {0} <password>".format(sys.argv[0]), file=sys.stderr)
    sys.exit(1)

password = sys.argv[1]
m = hashlib.sha256()
m.update(password.encode('utf-8'))
bd = m.digest()
pw_hash = base64.urlsafe_b64encode(bd).decode('ascii').replace('=', '')

print(pw_hash)
```

Here is an example using the script to generate a password hash for the password “foo”.

```
C:\bryanc\education\CSS475\rocket\rocket-schema>python get-password-hash.py foo
LCa0a2j_xo_5m0U8HTBBNBCLXBkg7-g-YpeiGJm564
```

These password hashes were used in the following SQL scripts.

```
-- =====
-- Creates Initial Users and Trivia Groups.
-- =====

-- Add the official team rocket users.
-- Keep static IDs from production.

INSERT INTO `User` (`UserId`,`Email`,`Alias`,`PasswordHash`)
VALUES (1,'castillo.bryan@gmail.com','bryan','NLPHDHR6e0gspxeIzozbdisld5hS90_A20Yt9U2zqw');

INSERT INTO `User` (`UserId`,`Email`,`Alias`,`PasswordHash`)
VALUES (4,'maridha@uw.edu','TechFatih','HF69uXo-j8ofUa7wHbJ4dWlv21GzsC1RU7JJjm2udPY');

INSERT INTO `User` (`UserId`,`Email`,`Alias`,`PasswordHash`)
VALUES (5,'trecewicklander@hotmail.com','Trece','su0YrgVwz8bkXk0Db3v1-PxpCaes0BknZYcYSSxLMoo');

INSERT INTO `User` (`UserId`,`Email`,`Alias`,`PasswordHash`)
VALUES (6,'fahaduwb@uw.edu','Fahad','Pqh6Vto4RLQg7CklrpIrxzHsFqT8Rny-r9rUmw5h05w');

-- This is the test user for grading.
-- The password is cuniculus
INSERT INTO `User` (`Email`,`Alias`,`PasswordHash`)
VALUES ('css451-tester@uw.edu','CSS451Tester','4RiFujva8vn3Q07wt6bciVd76sbVen7uT0nnop1RCaW');

-- Add some test users.

-- password: pw1
INSERT INTO User (Email, Alias, PasswordHash)
VALUES ('bob@gmail.com', 'bob', 'xZLfSoaT05Kt3JhCQC3fGYxjjqm-WJFu5uNzTh4xUvg');
```

```

-- password: pw2
INSERT INTO User (Email, Alias, PasswordHash)
VALUES ('joe@gmail.com', 'joe', 'k5FaCkv49jTLGFZJTdQwRHkRrmCf1QfdrZ2HEnMVbA');

-- password: pw3
INSERT INTO User (Email, Alias, PasswordHash)
VALUES ('zoe@gmail.com', 'zoe', 'Byqp6fudUWLkZdMyFTBGPK7NwbFWZ2_jBxmXzcEBeBY');

-- Create a couple groups.

INSERT INTO `Group` (GroupName, Description) VALUES ('Team Rocket', 'The best group');
INSERT INTO `Group` (GroupName, Description) VALUES ('UW Bothell', 'They will have jobs.');
```

```

INSERT INTO Group_Administrator VALUES (
  (SELECT UserId FROM User WHERE Email = 'bob@gmail.com'),
  (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

INSERT INTO Group_Administrator VALUES (
  (SELECT UserId FROM User WHERE Email = 'joe@gmail.com'),
  (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

INSERT INTO Group_Administrator VALUES (
  (SELECT UserId FROM User WHERE Email = 'zoe@gmail.com'),
  (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

-- Add Team Rocket "real" people to groups.
INSERT IGNORE INTO Group_Administrator
SELECT u.UserId, g.GroupId
FROM User u, `Group` g
WHERE
  u.Email IN (
    'castillo.bryan@gmail.com',
    'maridha@uw.edu',
    'trecewicklander@hotmail.com',
    'fahaduwb@uw.edu',
    'css451-tester@uw.edu');

-- =====
-- Creates some test questions and categories.
-- =====

-- Add some categories

INSERT INTO Category (CategoryName, GroupId)
VALUES (
  'Science',
```

```

    (SELECT GroupId FROM `Group` WHERE GroupName = 'Team Rocket')
);

INSERT INTO Category (CategoryName, GroupId)
VALUES (
    'P.E.',
    (SELECT GroupId FROM `Group` WHERE GroupName = 'Team Rocket')
);

INSERT INTO Category (CategoryName, GroupId)
VALUES (
    'Science',
    (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

INSERT INTO Category (CategoryName, GroupId)
VALUES (
    'Computer Science',
    (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

-- Add some questions.

SET @groupId = (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell');

INSERT INTO Question (Content, GroupId)
VALUES (
    'What is the chemical makeup of water?',
    (SELECT GroupId FROM `Group` WHERE GroupName = 'UW Bothell')
);

SET @qid = LAST_INSERT_ID();

INSERT INTO Answer
VALUES (1, @qid, 'H2O', True);

INSERT INTO Answer
VALUES (2, @qid, 'CO2', False);

INSERT INTO Answer
VALUES (3, @qid, 'H3O', False);

INSERT INTO Answer
VALUES (4, @qid, 'CH4', False);

INSERT INTO Question_Category
VALUES (
    (SELECT CategoryId FROM Category WHERE GroupId = @groupId AND CategoryName = 'Science'),
    @qid
);

```

Open Trivia Download And SQL Loading

For testing purposes the team loaded trivia questions and answers from opentb.com. 2 python scripts were used. The first one, open-trivia-downloader.py downloads trivia questions and saves them on disk as JSON files. The 2nd script, convert_to_mysql_sql.py, reads the questions, filters out some questions, and converts the data into insert statements for the RocketTrivia schema.

```
#-----
# Script: open-trivia-downloader.py
#
# Downloads trivia question data from opentb.com.
# Trivia questions will be written out as json files to the data directory.
#-----

import os
import http.client
import json
import datetime

class OpenTriviaDBClient:
    def __init__(self, endpoint):
        self.endpoint = endpoint
        self.token = None

    def get_questions(self):
        path = "/api.php?amount=50"
        if self.token != None:
            path = path + "&token=" + self.token
        return self.make_get_request(path)

    def set_token(self, token):
        self.token = token

    def acquire_token(self):
        response = self.make_get_request("/api_token.php?command=request")
        self.set_token(response["token"])

    def make_get_request(self, path):
        conn = http.client.HTTPSConnection(self.endpoint)
        try:
            conn.request("GET", path)
            r = conn.getresponse()
            if r.status != 200:
                raise Exception("Could not retrieve token.")
            content = r.read()
            payload = json.loads(content)
            if payload["response_code"] == 0:
                return payload
            elif payload["response_code"] == 1:
```

```

        return None
    else:
        raise Exception("Error received: " + str(payload["response_code"]))
finally:
    conn.close()

class Downloader:
    def __init__(self, endpoint, data_dir):
        self.data_dir = data_dir
        self.client = OpenTriviaDBClient(endpoint)

    def run(self, max_requests):
        self.client.acquire_token()
        if not os.path.exists(self.data_dir):
            os.makedirs(self.data_dir)

        t_stamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        req_count = 0
        while req_count < max_requests:
            questions = self.client.get_questions()
            if questions == None:
                break
            data_file = os.path.join(self.data_dir, "questions-" + t_stamp + "." + str(req_count + 1) +
".json")
            print("Writing content to: " + data_file)
            with open(data_file, "w") as fh:
                json.dump(questions, fh, sort_keys=True, indent=4)
            req_count = req_count + 1

def main():
    downloader = Downloader("opentdb.com", "data")
    downloader.run(70)

main()

#-----
# Script: convert_to_mysql_sql.py
#
# Filters questions downloaded from opentb.com and formats the data as
# insert statements suitable for the RocketTrivia schema.
#-----

import os
import json
import html

class Converter:
    def __init__(self, data_dir):
        self.data_dir = data_dir

    def run(self, sql_file):

```

```

# Group the questions by category
questions_by_cat = {}
for question in self.parse_questions():
    if not question["category"] in questions_by_cat:
        questions_by_cat[question["category"]] = []
    questions_by_cat[question["category"]].append(question)

with open(sql_file, "w") as sql_out:

    sql_out.write("SET autocommit = 0;\n")
    sql_out.write("BEGIN;\n")

    # Generate Cleanup
    sql_out.write("-- Clean up previous data\n\n")
    sql_out.write("DELETE FROM Answer\n")
    sql_out.write("WHERE questionId IN (\n")
    sql_out.write("  SELECT q.questionId\n")
    sql_out.write("  FROM Question q\n")
    sql_out.write("  JOIN `Group` g ON (g.groupName = 'Open Trivia' AND g.groupId = q.groupId)\n")
    sql_out.write(");\n")

    sql_out.write("DELETE FROM Question_Category\n")
    sql_out.write("WHERE questionId IN (\n")
    sql_out.write("  SELECT q.questionId\n")
    sql_out.write("  FROM Question q\n")
    sql_out.write("  JOIN `Group` g ON (g.groupName = 'Open Trivia' AND g.groupId = q.groupId)\n")
    sql_out.write(");\n")

    sql_out.write("DELETE FROM Question\n")
    sql_out.write("WHERE groupId IN (\n")
    sql_out.write("  SELECT g.groupId\n")
    sql_out.write("  FROM `Group` g\n")
    sql_out.write("  WHERE g.groupName = 'Open Trivia'\n")
    sql_out.write(");\n")

    sql_out.write("DELETE FROM Category\n")
    sql_out.write("WHERE groupId IN (\n")
    sql_out.write("  SELECT g.groupId\n")
    sql_out.write("  FROM `Group` g\n")
    sql_out.write("  WHERE g.groupName = 'Open Trivia'\n")
    sql_out.write(");\n")

    sql_out.write("DELETE FROM Group_Administrator\n")
    sql_out.write("WHERE groupId IN (\n")
    sql_out.write("  SELECT g.groupId\n")
    sql_out.write("  FROM `Group` g\n")
    sql_out.write("  WHERE g.groupName = 'Open Trivia'\n")
    sql_out.write(");\n")

```

```

sql_out.write("DELETE FROM `Group` WHERE groupName = 'Open Trivia';\n")
sql_out.write("COMMIT;\n")

# Create a group for the questions
sql_out.write("BEGIN;\n")
sql_out.write("-- Create the open trivia group\n")
sql_out.write("INSERT INTO `Group` (GroupName, Description) VALUES (\n")
sql_out.write(" 'Open Trivia',\n")
sql_out.write(" 'Questions from Open Trivia Database (https://opentdb.com/). All content is\n")
available under: (https://creativecommons.org/licenses/by-sa/4.0/legalcode).');\n")
sql_out.write("SET @groupId = LAST_INSERT_ID();\n")

sql_out.write("INSERT INTO Group_Administrator\n")
sql_out.write("SELECT u.userId, @groupId\n")
sql_out.write("FROM User u\n")
sql_out.write("WHERE u.Email in ('castillo.bryan@gmail.com', 'maridha@uw.edu',\n")
'trecekwicklander@hotmail.com', 'fahaduwb@uw.edu');\n")

seen_questions = set()

# Create the questions for each category.
for category in questions_by_cat.keys():
    sql_out.write("\n-- Generating content for category: {}\n".format(category))
    sql_out.write("INSERT INTO Category (CategoryName, GroupId) VALUES ({},\n")
@groupId);\n".format(self.to_sql_string(category)))
    sql_out.write("SET @categoryId = LAST_INSERT_ID();\n\n")

    for question in questions_by_cat[category]:
        content = self.to_sql_string(question["question"])

        if content in seen_questions:
            continue

        seen_questions.add(content)

        sql_out.write("INSERT INTO Question (Content, GroupId) VALUES ({},\n")
@groupId);\n".format(content))
        sql_out.write("SET @questionId = LAST_INSERT_ID();\n")
        sql_out.write("INSERT INTO Question_Category (CategoryId, QuestionId)\n")
VALUES(@categoryId, @questionId);\n")

        content = self.to_sql_string(question["correct_answer"])
        sql_out.write("INSERT INTO Answer VALUES(1, @questionId, {}, 1);\n".format(content))

        answerNum = 1
        for answer in question["incorrect_answers"]:
            answerNum = answerNum + 1
            content = self.to_sql_string(answer)
            sql_out.write("INSERT INTO Answer VALUES({}, @questionId, {},\n")
0);\n".format(answerNum, content))

```



```

        sql_out.write("COMMIT;\n")

def to_sql_string(self, s):
    ns = ""
    for c in s:
        if c == '\n':
            ns = ns + "\\n"
        elif c == '\r':
            ns = ns + "\\r"
        elif c == '\t':
            ns = ns + "\\t"
        elif c == '\\':
            ns = ns + "\\\\"
        elif c == '%':
            ns = ns + "\\%"
        elif c == '_':
            ns = ns + "\\_"
        elif c == "'":
            ns = ns + "'"
        elif c == '"':
            ns = ns + "\""
        else:
            ns = ns + c
    ns = ns + ""
    return ns

def has_basic_strings(self, result):
    def is_clean(s):
        for c in s:
            n = ord(c)
            if n >= 128 or n < 32:
                return False
        return True

    for a in result["incorrect_answers"]:
        if not is_clean(a):
            return False

    return is_clean(result["question"]) and \
           is_clean(result["correct_answer"]) and \
           is_clean(result["category"])

def parse_questions(self):
    files = os.listdir(self.data_dir)
    for f in files:
        if f.endswith(".json"):
            with open(os.path.join(self.data_dir, f), "r") as fh:
                data = json.load(fh)
                for result in data["results"]:

```

```

        if result["type"] != "multiple":
            continue
        if len(result["incorrect_answers"]) < 3:
            continue

        result["question"] = html.unescape(result["question"])
        result["correct_answer"] = html.unescape(result["correct_answer"])
        result["incorrect_answers"] = [html.unescape(a) for a in
result["incorrect_answers"]]
        result["category"] = html.unescape(result["category"])

        if not self.has_basic_strings(result):
            continue

    yield result

converter = Converter("data")
converter.run("open_trivia_questions.sql")

```

Game Generation REST API / Java Code

The following code snippets show parts of the Java code used to generate a Game. The Java code is exposed as REST API, which is called by the user interface. The first snippet of code shows the game controller which translated the REST input into calls against mappers. Mappers are classes which bind Java methods to SQL execution.

```

/**
 * Creates a new game.
 */
@RequestMapping(value = "/api/games", method = RequestMethod.POST)
public Game createGame(
    @RequestBody CreateGameRequest request,
    HttpServletRequest httpRequest,
    HttpServletResponse httpResponse) throws IOException, GeneralSecurityException
{
    logger.info("Request: " + request);

    AuthTokenIdentity identity = authTokenHandler.getAuthTokenIdentity(httpRequest);
    if (identity == null) {
        throw new AuthenticationException("Not logged in.");
    }

    try (SqlSession session = sqlSessionFactory.openSession()) {
        QuestionMapper qm = session.getMapper(QuestionMapper.class);
        GameMapper gm = session.getMapper(GameMapper.class);
    }
}

```

```

        // Get the questions randomized.
        List<Question> questions = qm.searchQuestionsByCategories(request.getCategoryIds(), true,
request.getMaxQuestions(), 0);
        List<Integer> questionIds = questions.stream().map(q ->
q.getQuestionId()).collect(Collectors.toList());

        // Get the answers in a random order.
        Map<Integer, List<Answer>> answers = qm.getAllQuestionAnswers(questionIds)
            .stream()
            .collect(Collectors.groupingBy(a -> a.getQuestionId()));

        // Create game
        gm.createGame(new Date(), request.getEndTime(), request.getMaxQuestionTime(),
identity.getUserId());
        int gameId = gm.getLastInsertId();

        // Add questions and answers
        for (int qIndex = 0; qIndex < questionIds.size(); qIndex++) {
            int questionId = questionIds.get(qIndex);
            gm.addGameQuestion(qIndex, gameId, questionId);
            List<Answer> questionAnswers = getRandomAnswers(answers.get(questionId));
            for (int aIndex = 0; aIndex < questionAnswers.size(); aIndex++) {
                gm.addGameQuestionAnswer(aIndex, qIndex, gameId, questionId,
questionAnswers.get(aIndex).getAnswerNumber());
            }
        }

        Game game = gm.getGame(gameId);
        session.commit();
        return game;
    }
}

```

The next snippet shows an example of one of the Mapper methods. MyBatis mappers allow the developer to annotate a Java method from an interface with a SQL statement. The parameters given to the method will be set as parameters on the sql statement. One additional feature that MyBatis also gives is SQL statement preprocessing. This can be seen in this method through the use of loops, tests, and choice blocks in the SQL statement. These parts are not SQL but allow you to run logic which is difficult in SQL. One of the most useful features we used this for was passing lists of ids dynamically to prepared statements.

```

public interface QuestionMapper {
    @Select({
        "<script>",
        "SELECT q.*",
        "FROM Question q",
        "WHERE",
        "  q.questionId IN (",

```

```

        "    SELECT DISTINCT qc.questionId",
        "    FROM Question_Category qc",
        "    WHERE",
        "        1 = 1",
        "        <if test='categoryIds != null'>",
        "            AND qc.categoryId IN",
        "                <foreach item='categoryId' index='categoryIds' collection='categoryIds' open='('
separator=', ' close=')'>",
        "                    #{categoryId}",
        "                </foreach>",
        "        </if>",
        "    )",
        "    <choose>",
        "        <when test='randomize'>",
        "            ORDER BY rand()",
        "        </when>",
        "        <otherwise>",
        "            ORDER BY q.questionId",
        "        </otherwise>",
        "    </choose>",
        "LIMIT #{maxResults} OFFSET #{offset}",
        "</script>"
    })
}

public List<Question> searchQuestionsByCategories(
    @Param("categoryIds") List<Integer> categoryIds,
    @Param("randomize") boolean randomize,
    @Param("maxResults") int maxResults,
    @Param("offset") int offset);

```

Game Play Simulation

The following code as used to simulate game play. This allowed the team to generate test game play data.

```

/**
 * This is a temporary API for simulating game play.
 */
@RequestMapping(value = "/api/game/{gameId}/play_simulate", method = RequestMethod.POST)
public void createGame(
    @PathVariable int gameId,
    HttpServletRequest httpRequest,
    HttpServletResponse httpResponse) throws IOException, GeneralSecurityException
{
    logger.info("Simulating play: " + gameId);

    AuthTokenIdentity identity = authTokenHandler.getAuthTokenIdentity(httpRequest);
    if (identity == null) {
        throw new AuthenticationException("Not logged in.");
    }
}

```

```

    }

    try (SqlSession session = sqlSessionFactory.openSession()) {
        GameMapper gm = session.getMapper(GameMapper.class);
        UserGameMapper ugm = session.getMapper(UserGameMapper.class);

        // Join the game if the user hasn't yet.
        UserGame userGame = ugm.getUserGame(identity.getUserId(), gameId);
        if (userGame == null) {
            logger.info("Joining game.");
            ugm.joinGame(identity.getUserId(), gameId);
        }

        // Get the previous answers the user made.
        List<UserGameQuestionAnswerPick> picks =
            ugm.getUserGameQuestionAnswerPicks(identity.getUserId(), gameId);
        Set<Integer> answeredQuestions = picks.stream().map(p ->
            p.getGameQNum()).collect(Collectors.toSet());

        Random rand = new Random();

        // Pick answers for the questions.
        List<GameQuestion> gameQuestions = gm.getGameQuestions(gameId);
        for (GameQuestion gq : gameQuestions) {
            if (!answeredQuestions.contains(gq.getGameQNum())) {
                logger.info("Need to answer: " + gq);

                // Make a random choice out of the answers.
                List<GameQuestionAnswer> possibleAnswers = gm.getGameQuestionAnswers(gameId,
                    gq.getGameQNum());
                GameQuestionAnswer answerChoice =
                    possibleAnswers.get(rand.nextInt(possibleAnswers.size()));
                logger.info("Choosing answer: " + answerChoice);

                UserGameQuestionAnswerPick pick = new UserGameQuestionAnswerPick();
                pick.setGameId(gameId);
                pick.setGameQANum(answerChoice.getGameQANum());
                pick.setGameQNum(answerChoice.getGameQNum());
                pick.setUserId(identity.getUserId());
                pick.setTimeChosen(new Date());
                pick.setTimeElapsed(1);

                ugm.setUserGameQuestionAnswer(pick);
            }
        }

        session.commit();
    }
}

```

The code above used Mappers to interact with SQL. The following shows one of the Mappers used to interaction with the database from Java.

```
public interface UserGameMapper {

    /**
     * Retrieves the game instance for a user id and game.
     */
    @Select({
        "SELECT ug.*",
        "FROM User_Games ug",
        "WHERE ug.userId = #{0} AND ug.gameId = #{1}"
    })
    public UserGame getUserGame(int userId, int gameId);

    @Insert({
        "INSERT INTO User_Games (userId, gameId, timeStarted)",
        "VALUES (#{0}, #{1}, CURRENT_TIMESTAMP)"
    })
    public void joinGame(int userId, int gameId);

    @Select({
        "SELECT p.*",
        "FROM User_Game_QA_Pick p",
        "WHERE p.userId = #{0} AND p.gameId = #{1}"
    })
    public List<UserGameQuestionAnswerPick> getUserGameQuestionAnswerPicks(int userId, int gameId);

    @Insert({
        "INSERT INTO User_Game_QA_Pick (userId, gameId, gameQNum, gameQANum, timeChosen, timeElapsed)",
        "VALUES (#{pick.userId}, #{pick.gameId}, #{pick.gameQNum}, #{pick.gameQANum}, #{pick.timeChosen}, "
        + "#{pick.timeElapsed})"
    })
    public void setUserGameQuestionAnswer(@Param("pick") UserGameQuestionAnswerPick pick);
}
```