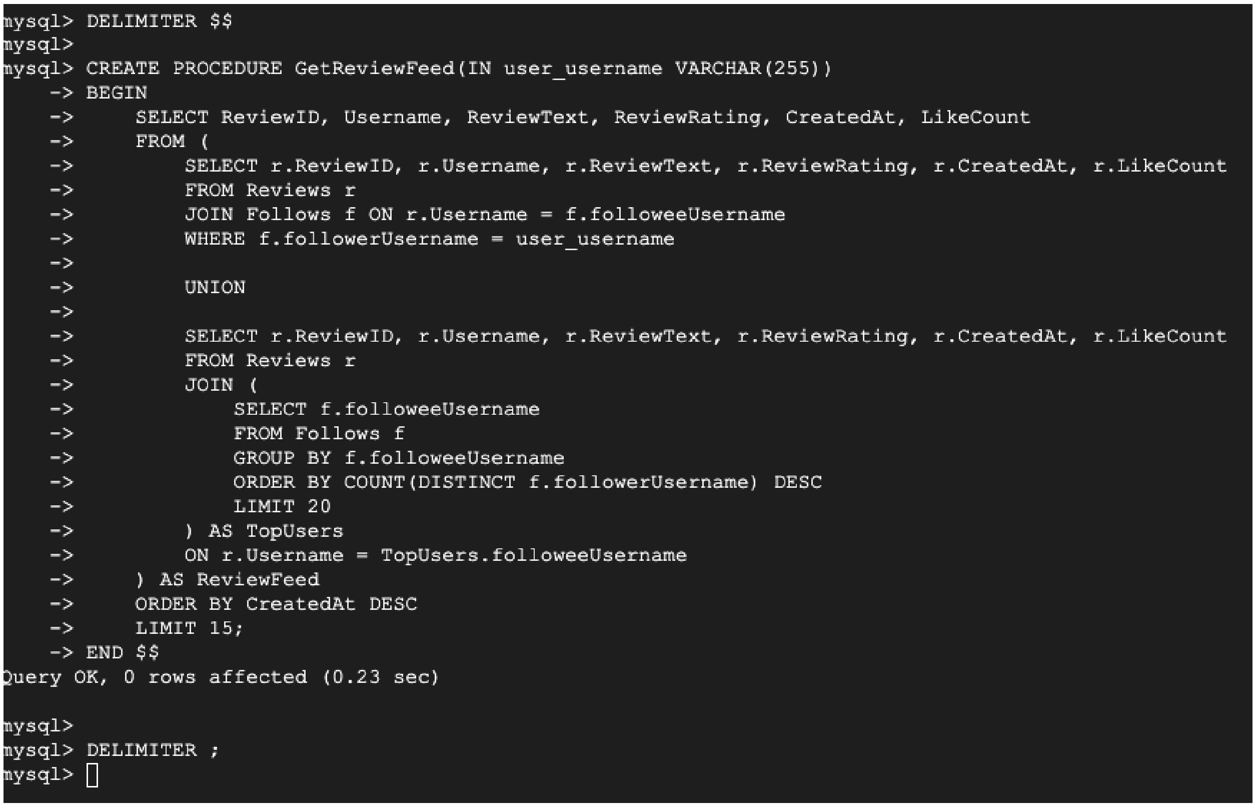
Advanced Queries

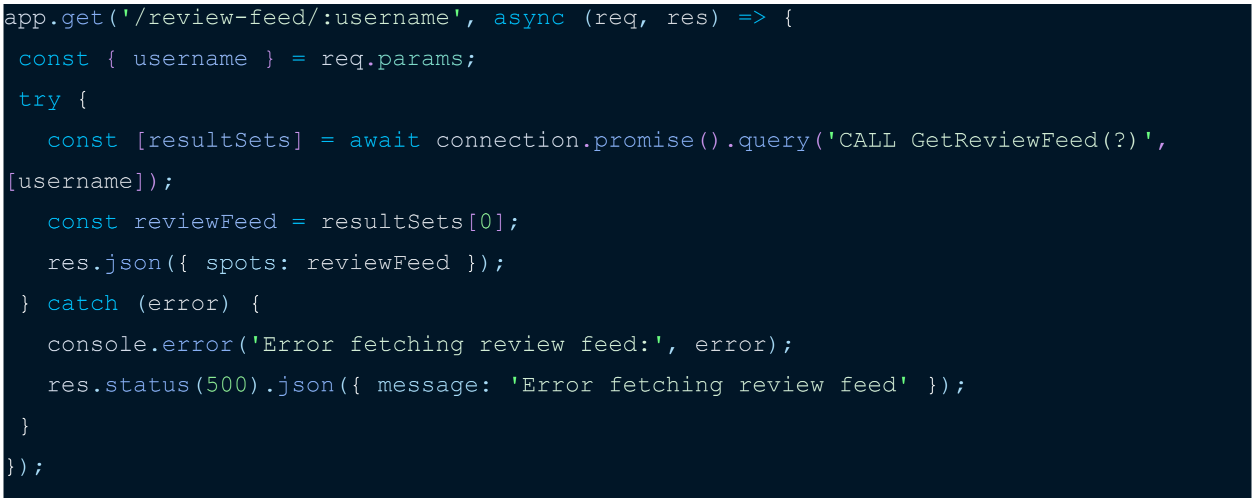
Procedures

- Query 1: Get recent reviews from users the current user follows and the top 20 users with the most followers.

Used in UserFeed on Browse homepage



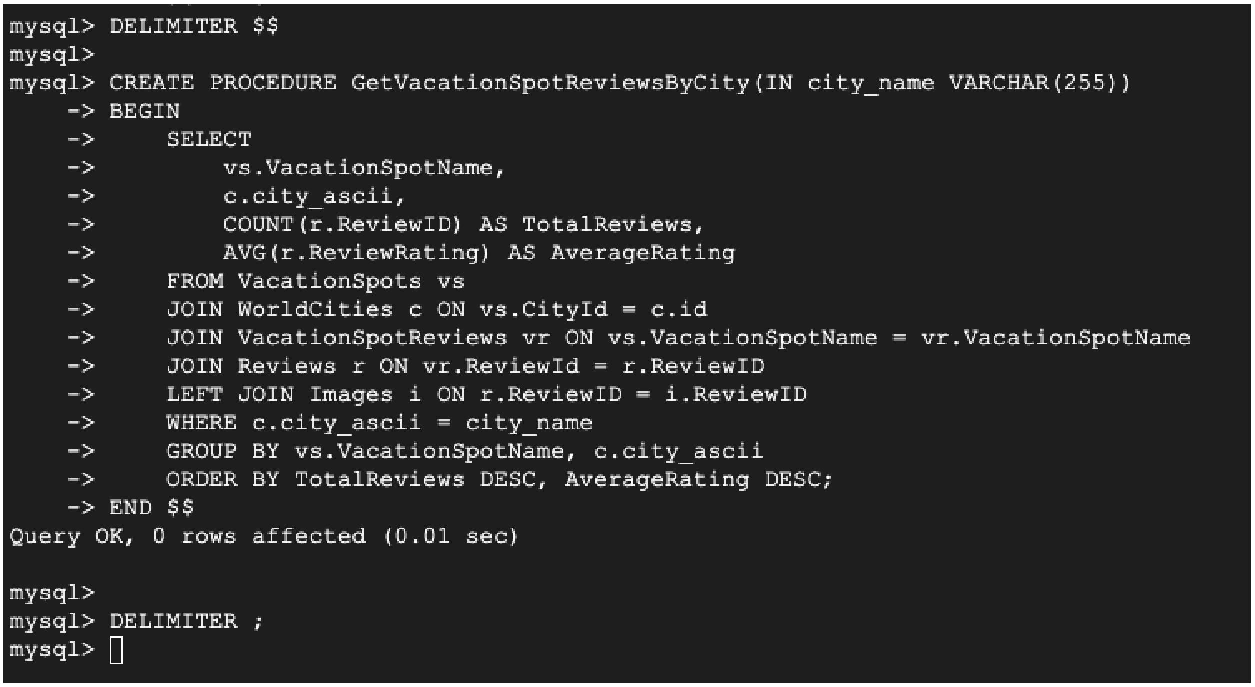
```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE GetReviewFeed(IN user_username VARCHAR(255))
    -> BEGIN
    ->     SELECT ReviewID, Username, ReviewText, ReviewRating, CreatedAt, LikeCount
    ->     FROM (
    ->         SELECT r.ReviewID, r.Username, r.ReviewText, r.ReviewRating, r.CreatedAt, r.LikeCount
    ->         FROM Reviews r
    ->         JOIN Follows f ON r.Username = f.followeeUsername
    ->         WHERE f.followerUsername = user_username
    ->
    ->         UNION
    ->
    ->         SELECT r.ReviewID, r.Username, r.ReviewText, r.ReviewRating, r.CreatedAt, r.LikeCount
    ->         FROM Reviews r
    ->         JOIN (
    ->             SELECT f.followeeUsername
    ->             FROM Follows f
    ->             GROUP BY f.followeeUsername
    ->             ORDER BY COUNT(DISTINCT f.followerUsername) DESC
    ->             LIMIT 20
    ->         ) AS TopUsers
    ->         ON r.Username = TopUsers.followeeUsername
    ->     ) AS ReviewFeed
    ->     ORDER BY CreatedAt DESC
    ->     LIMIT 15;
    -> END $$
Query OK, 0 rows affected (0.23 sec)

mysql>
mysql> DELIMITER ;
mysql>
```



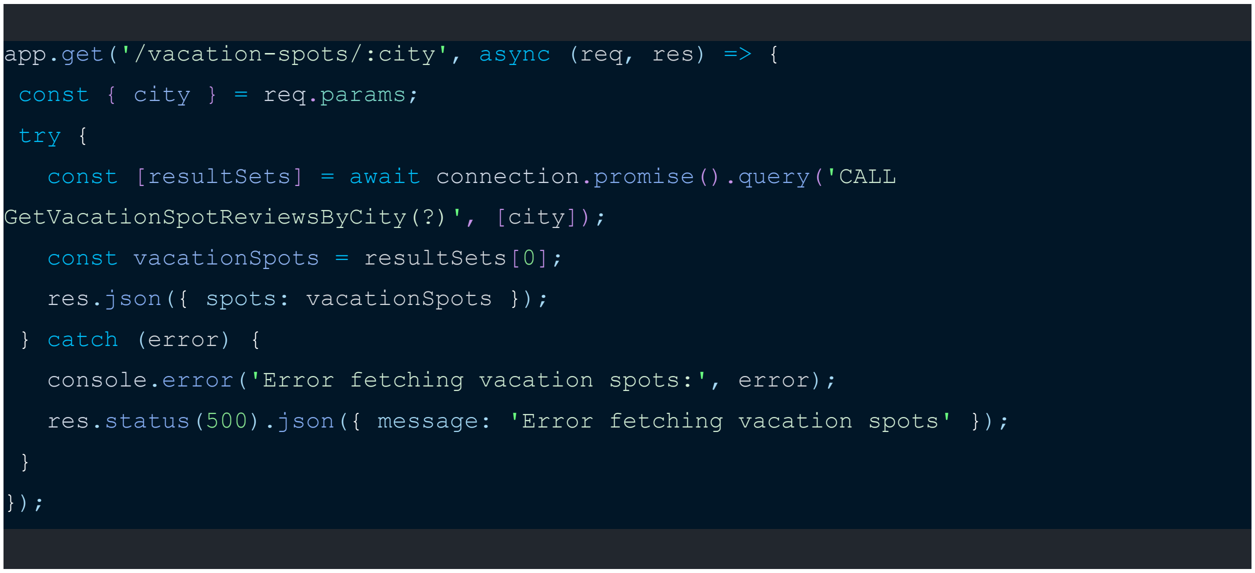
```
app.get('/review-feed/:username', async (req, res) => {
 const { username } = req.params;
 try {
   const [resultSets] = await connection.promise().query('CALL GetReviewFeed(?)',
[username]);
   const reviewFeed = resultSets[0];
   res.json({ spots: reviewFeed });
 } catch (error) {
   console.error('Error fetching review feed:', error);
   res.status(500).json({ message: 'Error fetching review feed' });
 }
});
```

- Query 2: find the most relevant vacation spots based on its popularity determined by number of reviews and average rating

Used in populating the most popular vacationSpots for a given city on Explore page



```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE GetVacationSpotReviewsByCity(IN city_name VARCHAR(255))
    -> BEGIN
    ->     SELECT
    ->         vs.VacationSpotName,
    ->         c.city_ascii,
    ->         COUNT(r.ReviewID) AS TotalReviews,
    ->         AVG(r.ReviewRating) AS AverageRating
    ->     FROM VacationSpots vs
    ->     JOIN WorldCities c ON vs.CityId = c.id
    ->     JOIN VacationSpotReviews vr ON vs.VacationSpotName = vr.VacationSpotName
    ->     JOIN Reviews r ON vr.ReviewId = r.ReviewID
    ->     LEFT JOIN Images i ON r.ReviewID = i.ReviewID
    ->     WHERE c.city_ascii = city_name
    ->     GROUP BY vs.VacationSpotName, c.city_ascii
    ->     ORDER BY TotalReviews DESC, AverageRating DESC;
    -> END $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> 
```



```
app.get('/vacation-spots/:city', async (req, res) => {
 const { city } = req.params;
 try {
   const [resultSets] = await connection.promise().query('CALL
GetVacationSpotReviewsByCity(?)', [city]);
   const vacationSpots = resultSets[0];
   res.json({ spots: vacationSpots });
 } catch (error) {
   console.error('Error fetching vacation spots:', error);
   res.status(500).json({ message: 'Error fetching vacation spots' });
 }
});
```

Transactions:

- Query 2: When defaultly viewing vacation spot reviews, selects top 3 reviews for a given vacation spot that have a like count greater than or equal to the average like count of all reviews for that vacation spot.

Also used in explore page to display reviews given the most popular vacationSpots within that city

```
app.get('/top-reviews', async (req, res) => {
 const spot = req.query.spot;
 if (!spot) {
   return res.status(400).json({ error: 'spot query-param is required' });
 }

 const conn = connection.promise();

 try {
   await conn.query('SET TRANSACTION ISOLATION LEVEL REPEATABLE READ');
   await conn.beginTransaction();

   const [rows] = await conn.query(
     `
     SELECT  r.ReviewID,
             r.Username,
             r.ReviewText,
             r.ReviewRating,
             r.CreatedAt,
             r.LikeCount,
             vsr.VacationSpotName,
             i.ImageURL
     FROM    Reviews             r
     JOIN    VacationSpotReviews vsr ON r.ReviewID = vsr.ReviewID
     LEFT    JOIN Images         i   ON r.ReviewID = i.ReviewID
     WHERE   vsr.VacationSpotName = ?
       AND   r.LikeCount >= (
               SELECT AVG(LikeCount)
               FROM   Reviews
               WHERE  ReviewID IN (
                       SELECT ReviewID
                       FROM   VacationSpotReviews
                       WHERE  VacationSpotName = ?
                     )
             )
     ORDER BY r.LikeCount DESC,
              r.CreatedAt DESC
     LIMIT 3
     `,
     [spot, spot]
   );

   await conn.commit();
   return res.json({ reviews: rows });

 } catch (err) {
   try { await conn.rollback(); } catch (_) {}
   console.error('top-reviews TX failed:', err);
   return res.status(500).json({ error: 'Failed to fetch top reviews' });
 }
});
```

- Query 4: Gets users favorite vacation spots that are also popular vacation spots (have greater than the average amount of likes of vacation spots)

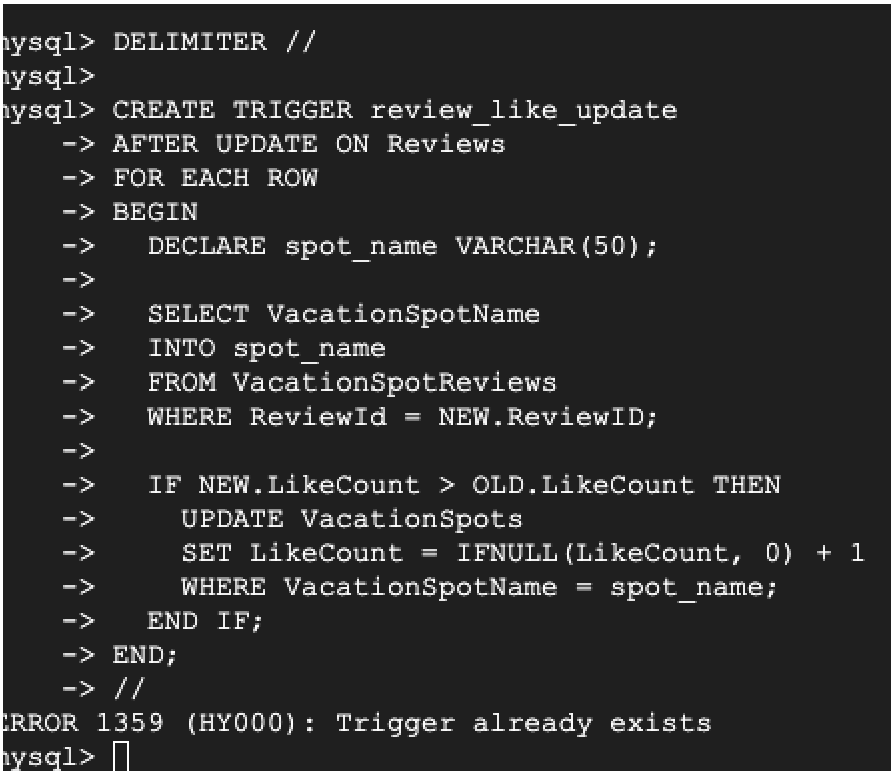
Used in log page to display favorite spots

```
app.get('/favorite-top-spots', async (req, res) => {
 const username = req.query.username;
 if (!username) {
   return res.status(400).json({ error: 'username query-param is required' });
 }

 const conn = connection.promise();

 try {
   await conn.query('SET TRANSACTION ISOLATION LEVEL REPEATABLE READ');
   await conn.beginTransaction();

   const [rows] = await conn.query(
     `
     SELECT fs.Username, fs.VacationSpotName, c.city, v.LikeCount
     FROM   FavoriteSpots fs
     JOIN   VacationSpots  v ON fs.VacationSpotName = v.VacationSpotName
     JOIN   WorldCities    c ON v.CityId          = c.id
     WHERE  fs.Username = ?

     INTERSECT

     SELECT fs.Username, fs.VacationSpotName, c.city, v.LikeCount
     FROM   FavoriteSpots fs
     JOIN   VacationSpots  v ON fs.VacationSpotName = v.VacationSpotName
     JOIN   WorldCities    c ON v.CityId          = c.id
     WHERE  v.LikeCount >= ( SELECT AVG(LikeCount) FROM VacationSpots )
     ORDER  BY LikeCount DESC
     LIMIT  15
     `,
     [username, username]
   );

   await conn.commit();
   return res.json({ topSpots: rows });

 } catch (err) {
   try { await conn.rollback(); } catch (_) {}
   console.error('favorite-top-spots TX failed:', err);
   return res.status(500).json({ error: 'Failed to fetch favourite spots' });
 }
});
```

Triggers

1. LikeUpdate

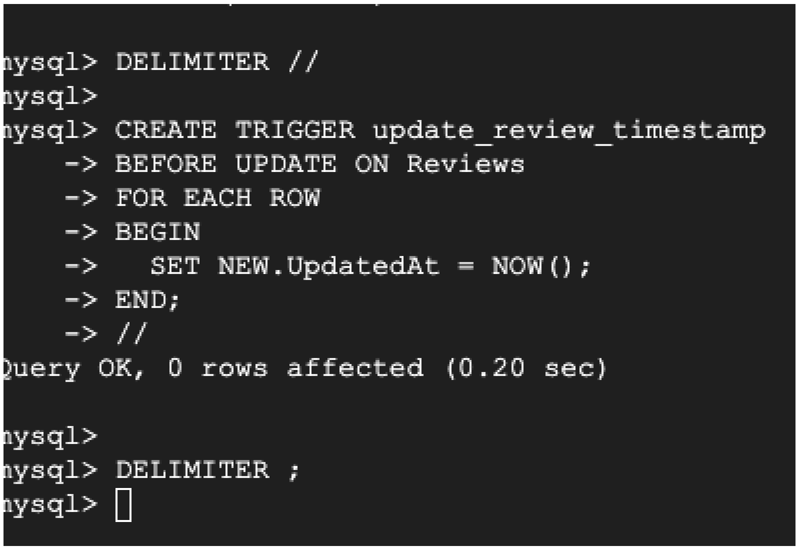
After a user likes a review for a VacationSpot, we will update the likeCount for the vacation spot associated with the review if it has improved it. This will help the application distinguish between “hot spots” for popular locations vs. less popular spots.



```
ysql> DELIMITER //
ysql>
ysql> CREATE TRIGGER review_like_update
    -> AFTER UPDATE ON Reviews
    -> FOR EACH ROW
    -> BEGIN
    ->   DECLARE spot_name VARCHAR(50);
    ->
    ->   SELECT VacationSpotName
    ->   INTO spot_name
    ->   FROM VacationSpotReviews
    ->   WHERE ReviewId = NEW.ReviewID;
    ->
    ->   IF NEW.LikeCount > OLD.LikeCount THEN
    ->     UPDATE VacationSpots
    ->     SET LikeCount = IFNULL(LikeCount, 0) + 1
    ->     WHERE VacationSpotName = spot_name;
    ->   END IF;
    -> END;
    -> //
RROR 1359 (HY000): Trigger already exists
ysql> 
```

2. Timestamp Update

If a user edits a review, we will update the timestamp to when it was last updated automatically. This ensures that our reviewTimestamps are as up to date as possible.

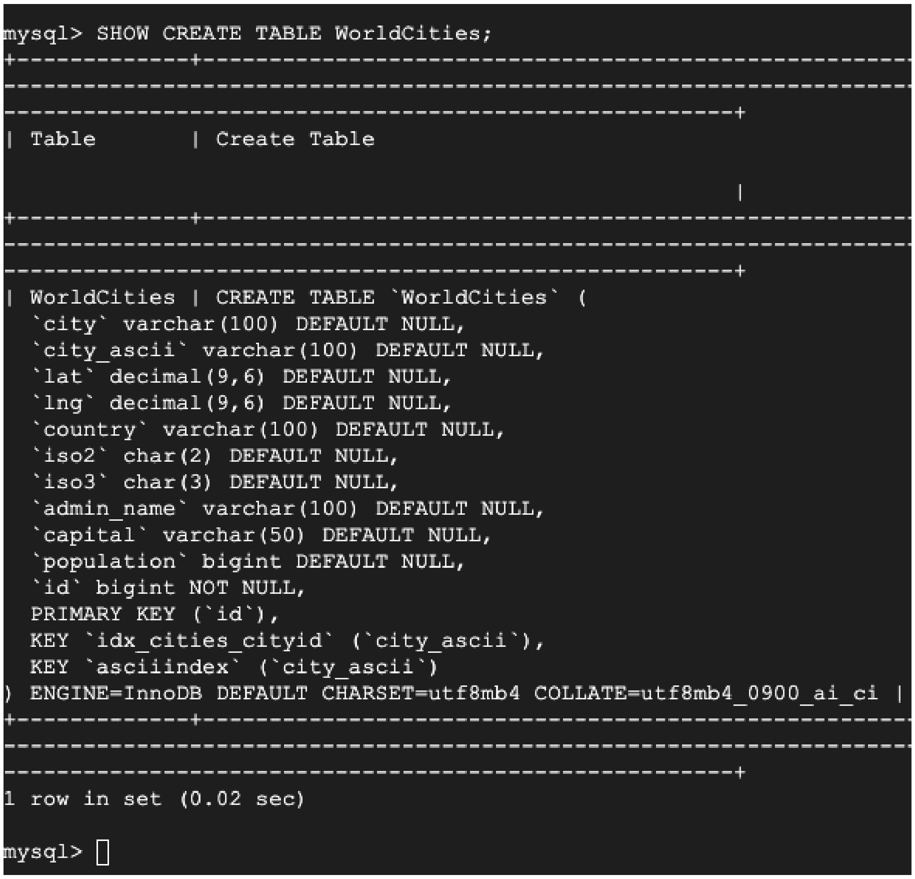


```
mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER update_review_timestamp
    -> BEFORE UPDATE ON Reviews
    -> FOR EACH ROW
    -> BEGIN
    ->   SET NEW.UpdatedAt = NOW();
    -> END;
    -> //
Query OK, 0 rows affected (0.20 sec)

mysql>
mysql> DELIMITER ;
mysql> 
```

Constraints

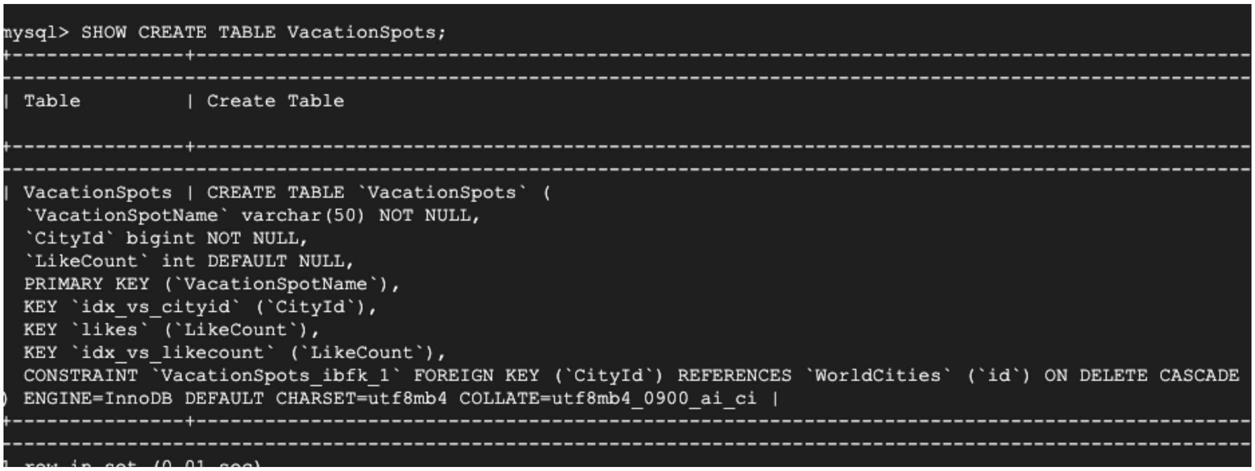
We have defined primary keys and foreign keys for the tables when initializing the DDL commands as shown below.

WorldCities -> ID is our primary key



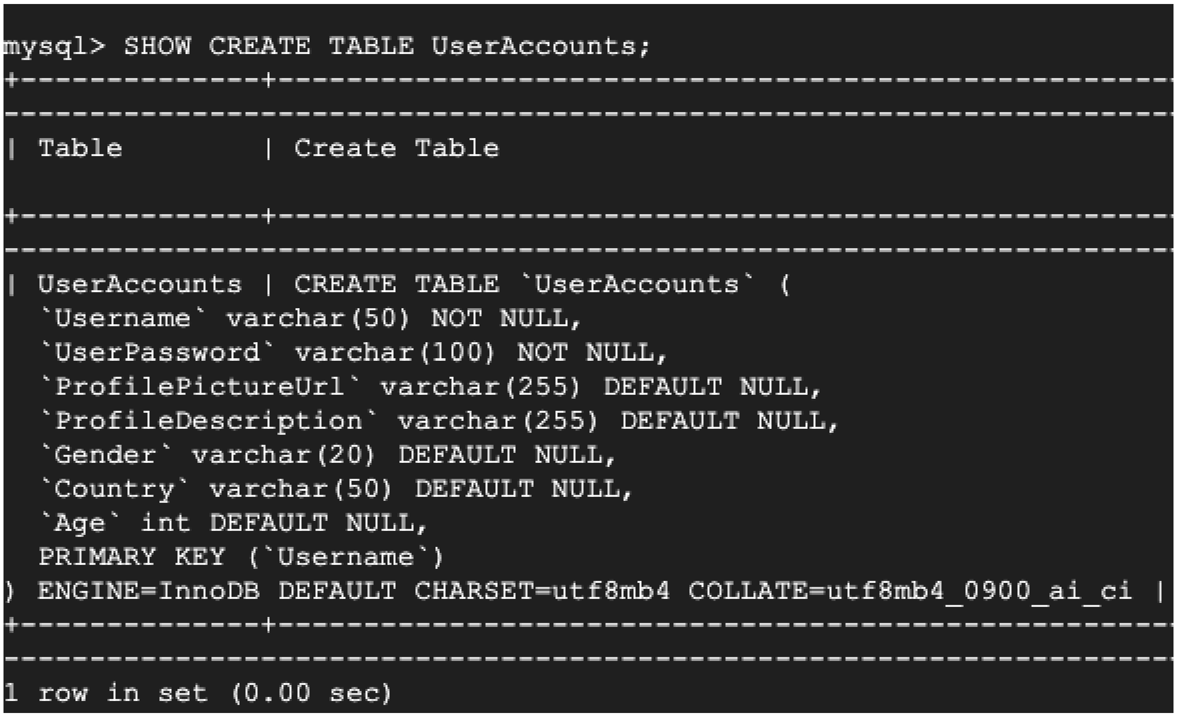
```
mysql> SHOW CREATE TABLE WorldCities;
+-------------+----------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------+
| Table       | Create Table

                                                        |
+-------------+----------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------+
| WorldCities | CREATE TABLE `WorldCities` (
  `city` varchar(100) DEFAULT NULL,
  `city_ascii` varchar(100) DEFAULT NULL,
  `lat` decimal(9,6) DEFAULT NULL,
  `lng` decimal(9,6) DEFAULT NULL,
  `country` varchar(100) DEFAULT NULL,
  `iso2` char(2) DEFAULT NULL,
  `iso3` char(3) DEFAULT NULL,
  `admin_name` varchar(100) DEFAULT NULL,
  `capital` varchar(50) DEFAULT NULL,
  `population` bigint DEFAULT NULL,
  `id` bigint NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_cities_cityid` (`city_ascii`),
  KEY `asciiindex` (`city_ascii`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------------+----------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------+
1 row in set (0.02 sec)

mysql> 
```

VacationSpots -> VacationSpotName is our primary key, has a foreign key CityID that references WorldCitiesID



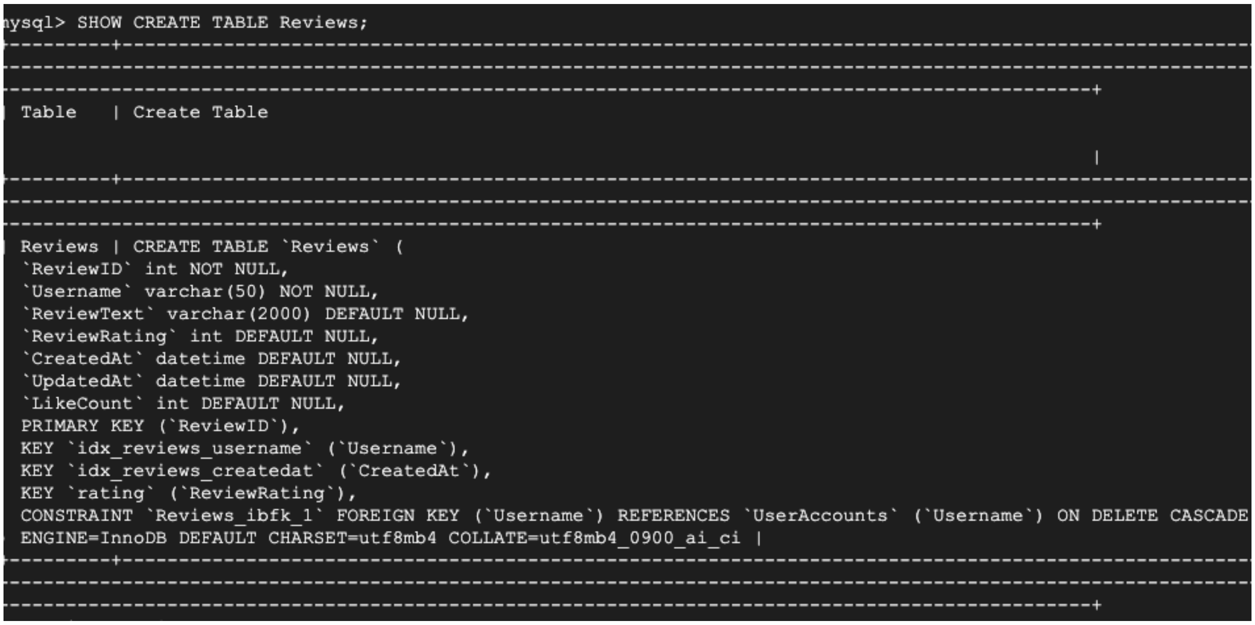
```
mysql> SHOW CREATE TABLE VacationSpots;
+---------------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
| Table         | Create Table

+---------------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
| VacationSpots | CREATE TABLE `VacationSpots` (
  `VacationSpotName` varchar(50) NOT NULL,
  `CityId` bigint NOT NULL,
  `LikeCount` int DEFAULT NULL,
  PRIMARY KEY (`VacationSpotName`),
  KEY `idx_vs_cityid` (`CityId`),
  KEY `likes` (`LikeCount`),
  KEY `idx_vs_likecount` (`LikeCount`),
  CONSTRAINT `VacationSpots_ibfk_1` FOREIGN KEY (`CityId`) REFERENCES `WorldCities` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

UserAccounts -> Username is our primary key that we force to be unique when users are creating.



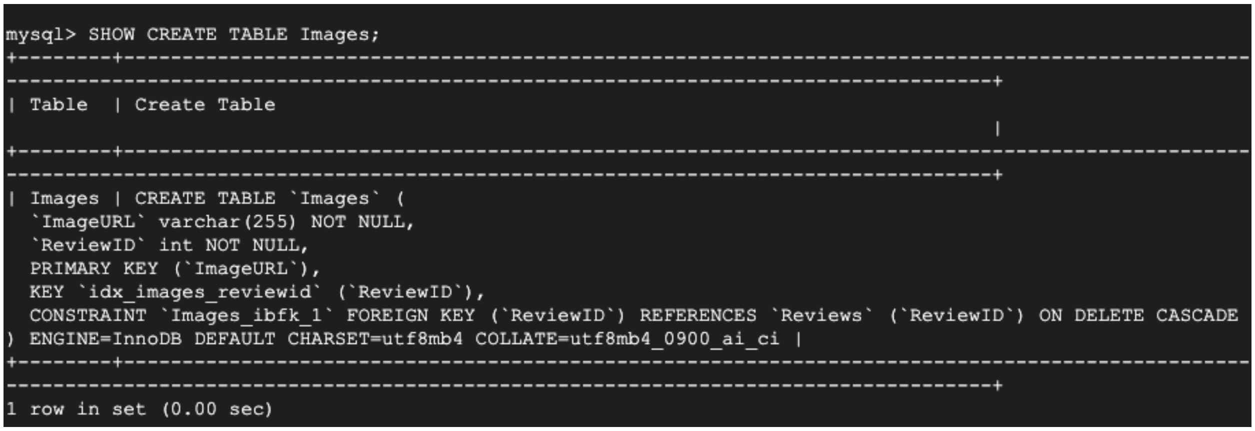
```
mysql> SHOW CREATE TABLE UserAccounts;
+--------------+-----------------------------------------------------
---------------------------------------------------------------------
| Table        | Create Table

+--------------+-----------------------------------------------------
---------------------------------------------------------------------
| UserAccounts | CREATE TABLE `UserAccounts` (
  `Username` varchar(50) NOT NULL,
  `UserPassword` varchar(100) NOT NULL,
  `ProfilePictureUrl` varchar(255) DEFAULT NULL,
  `ProfileDescription` varchar(255) DEFAULT NULL,
  `Gender` varchar(20) DEFAULT NULL,
  `Country` varchar(50) DEFAULT NULL,
  `Age` int DEFAULT NULL,
  PRIMARY KEY (`Username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------------+-----------------------------------------------------
---------------------------------------------------------------------
1 row in set (0.00 sec)
```

Reviews -> ReviewID is our primary key, Username is a foreign key which refers to the user who wrote the review, and references the UserAccounts Username attribute.



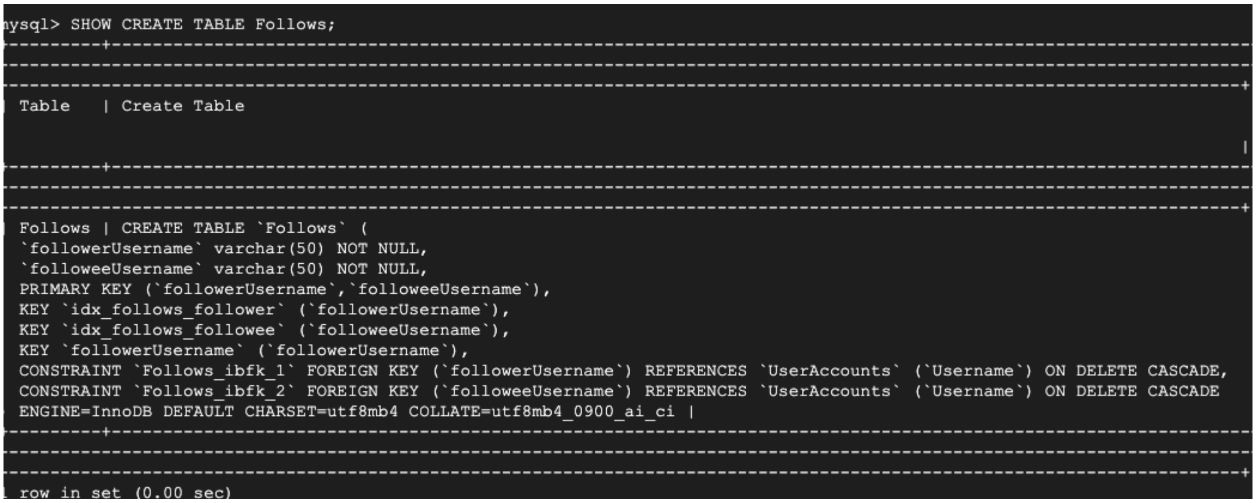
```
mysql> SHOW CREATE TABLE Reviews;
+---------+-------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| Table   | Create Table

                                                                                                 |
+---------+-------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| Reviews | CREATE TABLE `Reviews` (
  `ReviewID` int NOT NULL,
  `Username` varchar(50) NOT NULL,
  `ReviewText` varchar(2000) DEFAULT NULL,
  `ReviewRating` int DEFAULT NULL,
  `CreatedAt` datetime DEFAULT NULL,
  `UpdatedAt` datetime DEFAULT NULL,
  `LikeCount` int DEFAULT NULL,
  PRIMARY KEY (`ReviewID`),
  KEY `idx_reviews_username` (`Username`),
  KEY `idx_reviews_createdat` (`CreatedAt`),
  KEY `rating` (`ReviewRating`),
  CONSTRAINT `Reviews_ibfk_1` FOREIGN KEY (`Username`) REFERENCES `UserAccounts` (`Username`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------+-------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
```

Images -> ImageURL is our primary key since each image has its own source. It also has a foreign key ReviewID which references which review this image is associated with.



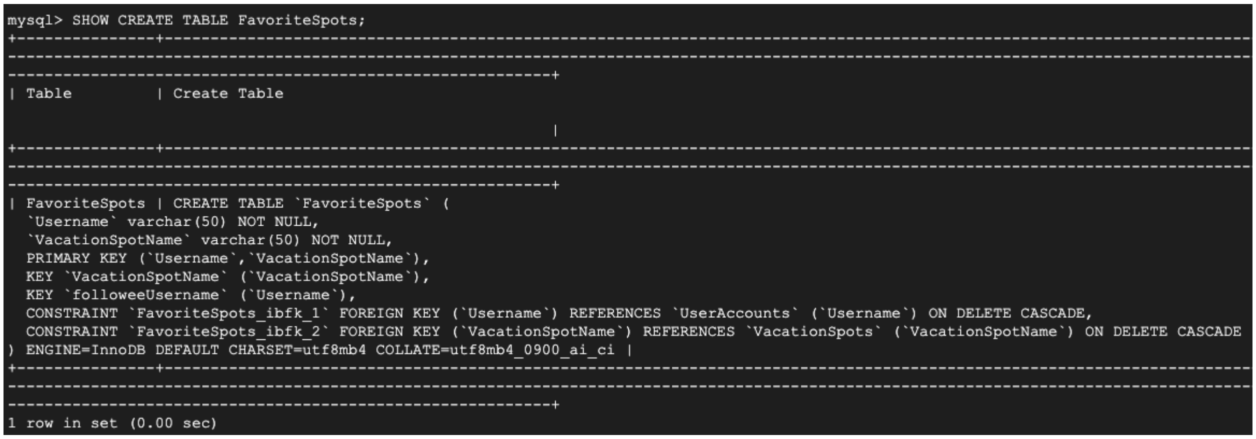
```
mysql> SHOW CREATE TABLE Images;
+--------+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| Table  | Create Table
                                                                                     |
+--------+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
| Images | CREATE TABLE `Images` (
  `ImageURL` varchar(255) NOT NULL,
  `ReviewID` int NOT NULL,
  PRIMARY KEY (`ImageURL`),
  KEY `idx_images_reviewid` (`ReviewID`),
  CONSTRAINT `Images_ibfk_1` FOREIGN KEY (`ReviewID`) REFERENCES `Reviews` (`ReviewID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+--------+-----------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Follows -> primary key on both the followee and follower since this is unique. There are foreign key constraints that both the followee and follower usernames must be in the UserAccounts table.



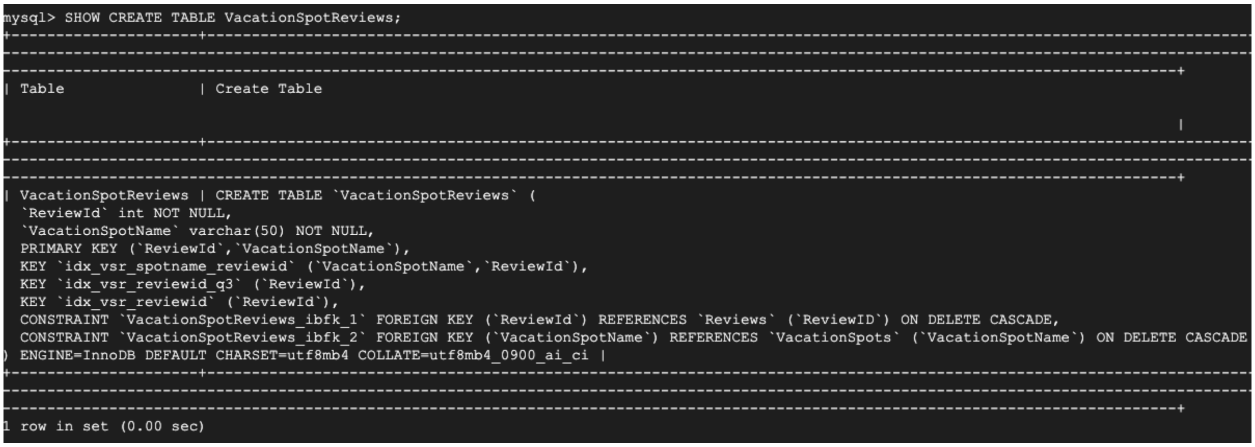
```
ysql> SHOW CREATE TABLE Follows;
---------+-----------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
 Table   | Create Table


                                                                                                                               |
---------+-----------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
 Follows | CREATE TABLE `Follows` (
  `followerUsername` varchar(50) NOT NULL,
  `followeeUsername` varchar(50) NOT NULL,
  PRIMARY KEY (`followerUsername`,`followeeUsername`),
  KEY `idx_follows_follower` (`followerUsername`),
  KEY `idx_follows_followee` (`followeeUsername`),
  KEY `followerUsername` (`followerUsername`),
  CONSTRAINT `Follows_ibfk_1` FOREIGN KEY (`followerUsername`) REFERENCES `UserAccounts` (`Username`) ON DELETE CASCADE,
  CONSTRAINT `Follows_ibfk_2` FOREIGN KEY (`followeeUsername`) REFERENCES `UserAccounts` (`Username`) ON DELETE CASCADE
  ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
---------+-----------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
 row in set (0.00 sec)
```

FavoriteSpots -> Primary Key on Username and VacationSpotname since each user is associated with their favorite spots. Foreign key references Username must be in User Accounts table and VacationSpotname must reference a vacationspotname from VacationSpots.



```
mysql> SHOW CREATE TABLE FavoriteSpots;
+---------------+------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| Table         | Create Table

                                                           |
+---------------+------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| FavoriteSpots | CREATE TABLE `FavoriteSpots` (
  `Username` varchar(50) NOT NULL,
  `VacationSpotName` varchar(50) NOT NULL,
  PRIMARY KEY (`Username`,`VacationSpotName`),
  KEY `VacationSpotName` (`VacationSpotName`),
  KEY `followeeUsername` (`Username`),
  CONSTRAINT `FavoriteSpots_ibfk_1` FOREIGN KEY (`Username`) REFERENCES `UserAccounts` (`Username`) ON DELETE CASCADE,
  CONSTRAINT `FavoriteSpots_ibfk_2` FOREIGN KEY (`VacationSpotName`) REFERENCES `VacationSpots` (`VacationSpotName`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------------+------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
1 row in set (0.00 sec)
```

VacationSpotReviews -> Primary key on both the reviewID and the VacationSpotname. Each review is unique to an ID and is assigned to a specific vacationspot. There are also foreign keys for reviewID to reference a reviewID from the Reviews table and VacationSpotname must reference a vacationspotname from VacationSpots.



```
mysql> SHOW CREATE TABLE VacationSpotReviews;
+---------------------+-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------+
| Table               | Create Table

                                                                                                                                      |
+---------------------+-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------+
| VacationSpotReviews | CREATE TABLE `VacationSpotReviews` (
  `ReviewId` int NOT NULL,
  `VacationSpotName` varchar(50) NOT NULL,
  PRIMARY KEY (`ReviewId`,`VacationSpotName`),
  KEY `idx_vsr_spotname_reviewid` (`VacationSpotName`,`ReviewId`),
  KEY `idx_vsr_reviewid_q3` (`ReviewId`),
  KEY `idx_vsr_reviewid` (`ReviewId`),
  CONSTRAINT `VacationSpotReviews_ibfk_1` FOREIGN KEY (`ReviewId`) REFERENCES `Reviews` (`ReviewID`) ON DELETE CASCADE,
  CONSTRAINT `VacationSpotReviews_ibfk_2` FOREIGN KEY (`VacationSpotName`) REFERENCES `VacationSpots` (`VacationSpotName`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+---------------------+-----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Project Reflection Report

Project Changes

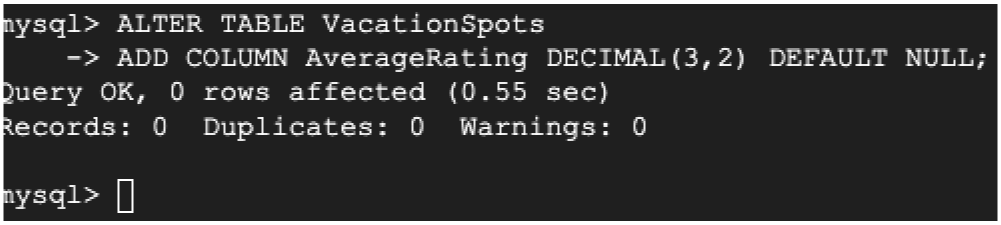
Since our stage 1 proposal submission, we made several changes for practicality. For example, we excluded some dataset information such as temperature which we were unable to match with the cities we found in other datasets (foreign key constraints). We also excluded information on specific activities within the cities since we populated some artificial data. Our images also were not completely implemented because we did not find a solution to host our image urls and get copyright for them. Lastly, while we were developing the frontend/backend application, we cut down for practicality to only US based cities and vacation spots. Otherwise, there were over 40,000 entries which made the traffic and lag on our website really heavy. However, all those entries are still in our database if we decide to improve traffic and latency with more optimized code. However, in general, we stuck with our project direction and did not pivot much from our original intent of a user-based vacation review platform.

Usefulness

We felt the application mainly accomplished the usefulness in tasks we set out in stage 1. We were able to create relevant data schemas that display all the information we would want in a vacation review platform. Users are able to view, create, edit, and delete reviews. They can view their logs and favorite spots. Most importantly, they are able to browse certain locations via a map and see relevant reviews in that area.

Data Schema/Source Changes

The main data schema we created in Stage 2 was used. We did not modify and create any additional tables or relationships. The only additions we changed was adding ratings, createdAt, and updatedAt for reviews, which we accidentally omitted. While creating the queries, we thought those were important metrics which are shown on reviews we would want to include. For the reviews, we generated some artificial data since we were unable to get real data.



```
mysql> ALTER TABLE VacationSpots
    -> ADD COLUMN AverageRating DECIMAL(3,2) DEFAULT NULL;
Query OK, 0 rows affected (0.55 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> 
```

Advanced Database Programs

Our advanced programs with procedures/transactions included the 4 queries:

```
-- 1. User Following Feed Retrieval
-- Get recent reviews from users the current user follows and the top 20 users with the most followers.
-- 2. Vacation Spot Review Page
-- When defaultly viewing vacation spot reviews, selects top 3 reviews for a given vacation spot that have a like count greater than or equal to the average like count of all reviews for that vacation spot.

-- 3. Top City Vacation Spots
-- When viewing city, find the most relevant vacation spots (including images) based on its popularity determined by number of reviews and average rating

-- 4. User Favorite Spot Retrieval
-- Gets users favorite vacation spots that are also popular vacation spots (have greater than the averge amount of likes of vacation spots)
```

This improves our application by providing relevant features for our social media feed and for helping users better explore cities/vacation spots. The first advanced program directly is the algorithm we use to populate a user's feed. The second advanced query helps populate vacation spot short summaries with the most descriptive and useful information based on user reviews. The third advanced query helps filter hot spots within cities. And lastly, the user favorite spot is used to better populate recommendations.

Technical Challenges

Ricky

A technical challenge I encountered was loading the data into the GCP MySQL Database. I thought it would be pretty straightforward, but there were lots of data error mismatches, as well as foreign key constraints. I had to manually check for what it looked like in the console, transform the data in my csv to better match if it wasn’t loaded properly, and then try again. The loading interface doesn’t give any troubleshooting messages if something goes wrong, so I had to diagnose those issues myself. It was a super tedious and time-consuming process

Andrew

One challenge that I faced while writing advanced queries was that the database had not yet been configured properly to allow me to test the accuracy of my queries. For example, some tables had not been created yet, while others did not have the necessary amount of tuples to return any data. Furthermore, some of the names of the tables did not match our predefined schema. To solve this, I wrote additional SQL queries to insert our existing data to populate necessary tables, ensuring that these data entries were realistic.

Yousef

A challenge that I faced was doing the frontend of the main project. While I thought that it would be easy, I never had much experience utilizing a database with a website, it was a very new process to me. Additionally, there was many small issues developing the frontend, from simple images not showing, to overlapping texts, which overall I overcame through trial and error, as well as researching my errors.

Owen

A challenge that I faced was creating the advanced queries for the project. It is difficult navigating the large datasets and creating a query that is both complex and relevant to the application. Furthermore, there was some complexity in adapting the queries to stored procedures and transactions alongside determining the accuracy of queries.

Future Work

Apart from a smoother front-end experience, we felt the application could gather larger sources of data that were more inclusive. Our application is limited in its scope since we were unable to load large amounts of data into the application without incurring costs in time to load and computation. If we had more resources, we would want to expand our platform to cover more than just several locations, but possibly nationwide or even global.

Another possible improvement we could make would be to create a mobile application that allows users to upload photos via their camera roll. This is a much more intuitive and easy process than users having to upload photos onto a computer and go onto a website to attach it. For simplicity purposes, we decided to go with a web platform since we were not that experienced in mobile application development.

Division of Labor

We felt the division of labor was fair and we worked well together. Each team member worked to their strengths and was responsible for their tasks. We communicated well and made sure to meet online when needed to collaborate. Below is a division of tasks

Met as a Team - Brainstormed ideas and drafted project proposal (Stage 1), Created UML Diagram (Stage 2)

Ricky - Setup GCP MySQL Server, Setup GCP VM, Initialized DDL Tables in GCP MySQL Database, Loaded Datasets into Database, Connected NodeJS server w/ GCP MySQL for CRUD for Reviews and Profile

Owen - Wrote Advanced Queries, Performed optimized indexing on queries, Stored Procedures, Constraints/Triggers

Andrew - Wrote Advanced Queries, Performed optimized indexing on queries, Transactions, Keyword Search Functionality

Yousef - Performed optimized indexing on queries, Frontend Hero Page Aesthetics, VacationSpot/City Pop Up Frontend