Brewmeister

Setup

Brewmeister runs locally and was built using MySQL, as reflected in the query and database logic. To run the application:

1. Run the beerinfo.sql script to create the tables, constraints, and under-the-hood table functionality.
2. Run the update.sql sql script to populate the database with tuples.
3. Run the Brewmeister application in Java. The main() function lives in BrewmeisterApplication.java.
4. Run python -m SimpleHTTPServer from the command line to create a local server for the application's website.
5. Ensure computer is connected to Internet, as some of the libraries used on the front end are not hosted locally.
6. Go to 0.0.0.0:8000 to interact with the application.

Code Used in the Application

Please see the Brewmeister compressed directory in the hand-in folder for all project code and files. Alternatively, all code is available on our team GitHub repository: https://github.com/TheSchnoo/Brewmeister.

Setup Script

Please see the beerinfo.sql script available in the Brewmeister compressed directory. The SQL script is also available on our team GitHub repository at https://github.com/TheSchnoo/Brewmeister/blob/master/beerinfo.sql.

What our Project Accomplished

From a learning perspective, we accomplished quite a bit with this project. Not only did it help us practice database design and structuring SQL queries, but it also taught us a lot about project management and the different pre-existing technologies available to speed up production.

For the back end of our application, we decided to use Spring's MVC module. It drastically reduced the amount of code we had to write for interacting with the front end, and provided a better way to organize our entire Java code base. For the front end we used AngularJS and Bootstrap. The former simplified the process of making requests to the back end and then populating the page with the retrieved data, while the latter provided an easier way to structure the layout of our html pages.

In terms of project management, we learned how important it was to properly plan all aspects out before starting on the actual implementation. We had to redesign the structure of our database several times once emergent requirements became more clear, and it ended up being very time consuming when this happened in the later stages of development. We did create documentation for all our APIs though, and this proved very useful when we needed to refer back to previous design decisions throughout the coding phase. We also used Trello to manage all the tasks that needed to be complete, which helped us coordinate the division of the workload and track what still needed to be completed.

Contrasting our Final Schema with our Submitted Schema

For the most part, we followed our planned schemas, adding a few additional attributes and functionalities for these attributes as needed. We found that these created more robust functionality and helped to streamline the control flow of our app. For instance, we added an additional attribute averageRating to the Beerinfo table and added SQL logic/triggers for recalculating and updating this averageRating field upon receiving new ratings. This allowed our app to calculate and provide average ratings functionality in an effective, efficient manner.

We also removed tables and fields that turned out to be redundant or not useful in functionality, such as FName in the Beerinfo table. Originally, we planned this field to represent the overall flavor name/profile of a beer. However, in development, we found that the other Beerinfo fields sufficiently described a beer's flavor profile, hence we removed the unnecessary field.

Furthermore, as a stretch goal, we'd hoped to add different types of users and account access. To this point, we've implemented customer and brewery users and in the future, we hope to add vendor users, as well. As such, we've retained the tables planned for vendor employees to help with extracurricular fleshing out of the app.

List of Queries Used

As a Regular User:

Creating a New Account:
INSERT INTO Customer VALUES('0', '<Input_CName>', '<Input_CPassword>');

Deleting an Account:
DELETE FROM Customer WHERE CName LIKE '<Input_CName> AND CPassword LIKE '<Input_CPassword>';

Logging In:
SELECT FROM Customer WHERE Cname like '<Input_CName>';

Searching for beers:
NOTE: beer searches can include from zero to six parameters relating to beers, allowing users to find beers with specific characteristics or to search more broadly. Optional parameters are indicated here with an opt_tag to indicate their status. Moreover, the code includes logic for dynamically adding AND, BETWEEN commas, =, <, and > as needed.

NOTE 2: The guts of this functionality provide a template for adding search parameters to any query beginning with "WHERE…". This template functionality is used as part of beer searches throughout the app.

Zero parameters:
SELECT * FROM Beerinfo

With parameters:
```
SELECT *
FROM Beerinfo
WHERE ABV >/</BETWEEN <opt_abv> AND <opt_abv2>
        AND BName LIKE '%<opt_BName>%'
        AND BType LIKE '%<opt_BType>%'
        AND avgRating > <opt_avgRating>
        AND ibu >/</BETWEEN <opt_ibu> AND <opt_ibu2>
        AND BreweryName LIKE '%<opt_BreweryName>%'
```

Finding All Vendors That Sell a Particular Beer:
SELECT bv.* FROM BeerVendor bv, BeerInStock bis WHERE bv.StoreId = bis.StoreId AND bis.BName LIKE '%<Input_BName>%'

Finding All Beers Sold By A Particular Vendor:
NOTE: this functionality makes use of the 'Searching for Beers' template query (details above), adding a new wrinkle for beers offered by a particular vendor. In doing so, the query presents logic for selecting beers from a particular vendor, then combines this with searching for beers functionality below. For brevity, we represent the portion which is identical to the 'Searching for Beers' queries with "searchBeers(beerSearchParameters)" where beerSearchParameters are the optional attributes used to narrow a beer search. For clarity, this portion of the query adds only the search parameters and logic for evaluating them. It does not include another "SELECT...FROM", etc.

SELECT bi.*
FROM Beerinfo bi, BeerVendor bv, BeerInStock bis
WHERE bi.BName = bis.BName AND bv.StoreId = bis.StoreId AND bv.StoreName LIKE '%<Input_StoreName>%' AND searchBeers(beerSearchParameters)

Getting the List of Beers with an Additional Attribute 'Stocked' for a Given Vendor
NOTE: This query returns a list of searched beers with an additional smallint/boolean attribute, 'Stocked', indicating whether a qualifying beer is in stock at a particular vendor. It makes use of the 'Searching for Beers' functionality (please see above for details) and for brevity, we will represent this portion with "searchBeers(beerSearchParameters)" where beerSearchParameters are the optional attributes used to narrow a beer search. For clarity, this portion of the query adds only the search parameters and logic for evaluating them, beginning with "WHERE…".

SELECT *, CASE WHEN
    (SELECT BName FROM BeerVendor bv, BeerInStock bis
    WHERE Beerinfo.BeerName = bis.BName AND bv.storeId = bis.storeId
    AND bv.storeId = <Input_storeId>) IS NULL THEN 0 ELSE 1 END AS Stocked
FROM Beerinfo
searchBeers(beerSearchParameters)

Seeing Other Users' Ratings for a Beer:
SELECT * FROM Rates WHERE BName LIKE '%bname%';

Submitting/Updating a Rating for a Beer:

For INSERT:

INSERT INTO Rates VALUES(CID, '<Input_BName>', <Input_BRate>, '<Input_Review>');

For UPDATE:

```
UPDATE Rates SET BRate = brate WHERE BName LIKE '%bname%' AND CID=cid;
UPDATE Rates SET Review = review WHERE BName LIKE '%bname%' AND CID=cid;
```

For INSERT/UPDATE:

A trigger is activated which updates the AvgRating field of the updated/added beer review's beer in the beerinfo table:

```
UPDATE BeerInfo
        SET AvgRating =
                (SELECT AVG(Rates.BRate)   FROM Rates WHERE BName=beername)
        WHERE beername = BName;
```

Get N Top Rated Beers:

```
SELECT * FROM Beerinfo ORDER BY AvgRating DESC LIMIT + <Input_N>
```

Get the Most Rated Beer:

```
SELECT b1.*, rates_count
FROM BeerInfo b1
JOIN (SELECT b2.BName, COUNT(r1.BName) AS rates_count
        FROM BeerInfo b2, rates r1
        WHERE b2.BName = r1.BName
        GROUP BY b2.BName) AS CountRatings
ON b1.BName = CountRatings.BName
WHERE rates_count =
        (SELECT MAX(rates_count)
        FROM (SELECT b2.BName, COUNT(r1.BName) AS rates_count
                FROM BeerInfo b2, rates r1
                WHERE b2.BName = r1.BName
                GROUP BY b2.BName) AS CountRatings)
```

Find Top Four Highest Rated Beers That the User Has Not Yet Reviewed

NOTE: This is functionality helps signed-in users to find highly-rated beers they have not yet tried. For users who have not reviewed any beers or if the user has not logged in, this query returns the top four highest rated beers in the database as new beer suggestions.

```
SELECT bi.* FROM Beerinfo bi WHERE NOT EXISTS
        (SELECT Beerinfo.*, r.cid FROM Beerinfo, Rates r, Customer c
```

WHERE r.BName = bi.BName AND r.CID = c.CID AND c.CID = <Input_CID>)
ORDER BY bi.AvgRating DESC LIMIT 4

As a Beer Vendor User:

Creating a New Account:
INSERT INTO BeerVendor VALUES('0', '<Input_StoreName>', '<Input_SPassword>');

Logging In:
SELECT FROM BeerVendor WHERE StoreName like '<Input_StoreName>';

Adding a Beer to my Inventory:

```
INSERT INTO beerinstock(BName,StoreId)
VALUES ('<Input_bname>','<Input_storeiD>')
from beerinfo;
```

Removing a Beer in my Inventory:

```
DELETE FROM beerinstock
WHERE StoreId = storeid and BName like '%bname%';
```

As a Brewery User:

Adding a New Beer to the System:
INSERT INTO BeerInfo(BName, BType, IBU, ABV, Description, BreweryName)
VALUES ('<Input_bname>','<Input_btype>', '<Input_ibu>, '<Input_abv>,
'<Input_description>','<Input_brewery>');

Removing a Beer from Production:
Breweries cannot actually remove beers from the system because a) Beers will persist on shelves and in fridges even after production ends, and b) We want the customer rating of the beer to continue to help the refining of our beer recommendations for them. Here is what a Brewery does when they stop making a beer:

```
UPDATE beerinfo
```

SET Brewed=false
WHERE BName like ‘%bname%’;

Updating a Beer’s Description:
UPDATE Beerinfo
SET Description=<Input_Description>
WHERE BName LIKE ‘%BName%’

List of Functional Dependencies

BName → BType, IBU, ABV, Description, BreweryName, Brewed, AvgRating
Unique beer names indicate combinations of beer type, flavor profile (including IBU, ABV, and a description), brewing status, and average rating.

CID, BName → Stars, WrittenReview
Unique combinations of a customer ID numbers and beer names identify user reviews of specific beers, including a rating out of five and a written review.

EmpID → EmpName, StoreID
Each employee of a vendor has a name, store ID number for the vendor they work for, and a unique employee ID number

CID → CName
Customer names are based on a customer’s ID number.

CName → CID
1to1 link between CNAME and CID, both are UNIQUE

StoreID → StoreName, Address
Each vender has a name, an address, and a unique StoreID

StoreName, Address → StoreID
1to1 link between (StoreName,Address) and StoreID