Project : Phase 2

ARPAN SHAH, Rochester Institute of Technology
DIPTANU SARKAR, Rochester Institute of Technology
LIPISHA NITIN CHAUDHARY, Rochester Institute of Technology
RITABAN BHATTACHARYA, Rochester Institute of Technology

This document presents the Phase 2 of the Project.

CCS Concepts: • **Information systems → Data management systems**;

Additional Key Words and Phrases: big data, relational model, postgresql, computer science, database, database management, normalization, nosql, mongodb

1 INTRODUCTION

This report provides the following based on the requirements of Phase 2:

- Proposes a document-oriented model for the dataset selected for Phase 1 and comparison with the relational model proposed in Phase 1.
- Provides a program which issues 7 interesting SQL queries over the relational model proposed in Phase 1. Then the tables are indexed to optimize the result of the queries.
- Provides all the functional dependencies and normalization with respect to the relational model proposed in Phase 1.

2 DOCUMENT-ORIENTED MODEL FOR THE DATASET

Fig. 1. Document-Oriented Model

2.1 Description

Fig. 1 represents the model proposed for the document-oriented model for the dataset selected for Phase 1. The model depicted above provides description about two collections – beers and breweries. In the beers collections the field _id uniquely describes an individual beer, we have merged the review table from our original schema with the beer collection. The review field has similar nested data objects which gives information about the reviews provided for the beer for which the specific document is created. The brewery field in the beer collection refers to the _id in the breweries collection, this shows the relationship between the beer and breweries collection.

2.2 Comparison

For the Project Phase 1, we had proposed a comprehensive relational model which consisted data of the Brewery and Reviews separate tables, along with the data for beers segregated in such a way that it was robust enough to independently define a particular aspect of the beer, for example we had defined relations style and availability with each having their unique id mapped to the ids in the Beer table.

Authors' addresses: Arpan Shah, as6999@rit.edu, Rochester Institute of Technology; Diptanu Sarkar, ds9297@rit.edu, Rochester Institute of Technology; Lipisha Nitin Chaudhary, lc2919@rit.edu, Rochester Institute of Technology; Ritaban Bhattacharya, rb5344@rit.edu, Rochester Institute of Technology.

Looking at how document oriented models provides us a compact yet complete way of defining the tables, we came up with the model described in *Fig. 1*, which collects all the data related to beers in a single document (collection) called Beers. If further elucidating this point, we had three major tables – Beers, Brewery and Reviews, in which the Reviews table had all the reviews pertaining to a particular beer, given by multiple users. Taking into consideration how these both tables – Beers & Reviews - supplied information about the same object, we merged the data from both the tables based on the beer_id, to further condense the data into a single document.

The advantage of this was that whenever we wanted to get any data related to a single beer we would precisely provide its id and achieve all the data along with its reviews in one go.

Fig. 2 depicts how we intend to view the data.

Fig. 2. Fetching Records from the document-oriented model

3 INTERESTING SQL QUERIES

3.1 Queries

The program for the queries is provided along with this report in *interesting_queries.py*

The queries are created based on finding interesting results which can be helpful to the viewer and also enable us to provide interesting joins and other functions. We provide a brief description of the goal of each query and how they were implemented.

- *Top high abv beers we wish could be brewed again*
  To find out the beers, the beer table is joined with the review table on the beer id. Then to find beers not in service anymore, the retired column of beer is checked to be true. For high abv and top beers, conditions are added for abv greater than 10, and average of all the overall ratings of a beer (found out using group by beer id) greater than 4.8.
- *Top 5 beers we must try based on score and rating*
  To find out the beers, the beer table is joined with the review table on the beer id. Then to find beers still in service, the retired column of beer is checked to be false. The average overall rating and average score of the beer is calculated after grouping the records using beer id. The score of a beer review is calculated as the average rating of look, smell, taste and feel. Then the records having average rating and average score greater than 4.5 are displayed.
- *Limited beers that are available in the USA with high abv*
  To find out beers brewed in United States, the beer table is joined with brewery table and availability table on brewery id and availability id respectively and then, conditions are added for brewery country is "US" and availability is limited. For high abv, abv from beer is checked to be greater than 10. The beer table is also joined wih review table on beer id and the overall ratings are grouped for beer id and brewery id to order the records based on descending order of the ratings.
- *Well-liked available beers that are home-brewed outside of the USA*
  To find out homebrewed beers that are still in service, the beer table is joined with brewery table on brewery id which is then joined with brewery_type table on brewery id for brewery country not equal to "US" and beer is not retired. The records are grouped on beer id and brewery id to aggregate all the types of that beer and brewery. Then if among the types, "Homebrew" exists, the records are displayed.

- *Top-rated(based on look, smell, taste and feel) beers, which are available Year-round in NY*
  To find out Year-Round available beer of New York, the beer table is joined with brewery table and availability table on brewery id and availability id respectively and then, conditions are added for beer not retired, brewery country is "US", brewery state is "NY" and availability is "Year-round". The beer table is also joined wih review table on beer id and the overall ratings are grouped for beer id and brewery id to order the records based on descending order of the average score that is calculated as average rating based on look, smell, taste and feel.
- *US breweries that brew more than 1000 top rated beer*
  To find out breweries in United States with more than 1000 top rated beers, the beer table is joined with brewery table on brewery id and then joined with reviews on beer id for brewery country as "US". The results are grouped on beer id to count all the overall ratings in review. The count is based on the case that if the rating is greater than 4.5, then it will be counted. The records are displayed based on the descending count and for the count being greater than 1000.
- *Style of beer that has the most number of beers registered*
  For style of beer, the style table is joined with beer table on style id. The beer id are grouped on the style id and the records are order in descending order of the count of beer id for a particular style. The record is limited to 1 to find the top style of beer.

3.2 Indexing the Queries

The indices for the queries are provided along with this report in *index.sql*.

The indices are added on the queries based on their conditions and constraints so as to speed up the query execution and minimize the time taken to execute the queries.

Query	Before Indexing	After Indexing
Top high abv beers we wish could be brewed again	15 seconds	8 seconds
Top 5 beers we must try based on score and rating	32 seconds	21 seconds
Limited beers that are available in the USA with high abv	321 miliseconds	156 miliseconds
Well-liked available beers that are home-brewed outside of the USA	500 miliseconds	432 miliseconds
Top-rated(based on look, smell, taste and feel) beers, which are available Year-round in NY	700 miliseconds	503 miliseconds
US breweries that brew more than 1000 top rated beer	20 seconds	12 seconds
Style of beer that has the most number of beers registered	193 miliseconds	142 miliseconds

Fig. 3. Comparison of the runtime of queries before and after indexing

4 FUNCTIONAL DEPENDECIES AND NORMALIZATION

The functional dependencies and normalization are implemented using a program *FunctionalDependency.java* provided along with the report.

For finding functional dependencies, a max of two attributes on the LHS were used. Any trivial dependencies in the form of a -> a or ab -> c when a -> c already exists were pruned out.

Below contains the functional dependencies for each table in our database. It also contains the computation of the candidate keys, canonical cover, and 3NF conversion for each table.

4.1 Brewery

Functional dependencies -

- id -> name
- id -> city
- id -> state
- id -> country
- name, city -> country

Candidate Keys -

- Core: id
- Exterior: name, city, state, country
- Closure of core: {id, name, city, state, country}
- Candidate key: id

Canonical Cover -
We can remove the functional dependency [name, city -> country] because these can all be determined with the functional dependencies with id on the LHS.
F(c): id -> name, city, state, country
3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(id, name, city, state, country)
r1 contains a candidate key for Brewery and there are no redundant relations, so r1 is the only relation needed.

4.2 Style

Functional dependencies -

- id -> style
- style -> id

Candidate Keys -

- Core: N/A
- Exterior: id, style
- Closure of core: {}
- Closure of exterior: {id, style}
- Candidate key: id, style

Canonical Cover -
- id -> style
- style -> id
3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(id, style)
- r2(style, id)
r1 contains a candidate key for Style and r2 is a rundandant relation to r1, so we remove it. Thus, r1 is the only relation needed.

4.3 Type

Functional dependencies -

- id -> type
- type -> id

Candidate Keys -

- Core: N/A
- Exterior: id, type
- Closure of core: {}
- Closure of exterior: {id, type}
- Candidate key: id, type

Canonical Cover -
- id -> type
- type -> id

3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(id, type)
- r2(type, id)

r1 contains a candidate key for type and r2 is a rundandant relation to r1, so we remove it. Thus, r1 is the only relation needed.

4.4 Availability

Functional dependencies -

- id -> availability
- availability -> id

Candidate Keys -

- Core: N/A
- Exterior: id, availability
- Closure of core: {}
- Closure of exterior: {id, availability}
- Candidate key: id, availability

Canonical Cover -
- id -> availability
- availability -> id

3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(id, availability)
- r2(availability, id)

r1 contains a candidate key for availability and r2 is a rundandant relation to r1, so we remove it. Thus, r1 is the only relation needed.

4.5 Brewery_Type

Functional dependencies -

- None

Candidate Keys -

- Core: N/A
- Exterior: brewery, type
- Closure of core: {}
- Closure of exterior: {brewery, type}
- Candidate key: brewery, type

Canonical Cover -

- None

3NF Conversion-
Create relations r(ab) for all [a -> b]
- None
Since there is no relation containing a candidate key, we must create one:
- r1(brewery, type)
r1 contains a candidate key for Brewery_Type and is the only relation needed.

4.6 Beer

Functional dependencies -

- id -> name
- id -> brewery
- id -> style
- id -> availability
- id -> abv
- id -> retired

Candidate Keys -

- Core: id
- Exterior: name, brewery, style, availability, abv, retired
- Closure of core: {id, name, brewery, style, availability, abv, retired}
- Candidate key: id

Canonical Cover -
We can consolidate the functional dependencies in F to one functional dependency.
F(c): id -> name, brewery, style, availability, abv, retired
3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(id, name, brewery, style, availability, abv, retired)
Since the relation contains a candidate key, we do not need to create others.
There are no duplicates or redundant relations to remove.
r1 contains a candidate key for Beer and is the only relation needed.

4.7 Reviews

Functional dependencies -

- None (However with 3 on LHS, [beer, username, review_date] -> every other attribute)

Candidate Keys -

- Core: {}
- Exterior: beer, username, review_date, review, look, smell, taste, feel, overall
- Closure of core: {}
- Closure of exterior: {beer, username, review_date, review, look, smell, taste, feel, overall}
- Candidate key: beer, username, review_date

Canonical Cover -
Using the functional dependencies with 3 attributes on LHS-
F(c): beer, username, review_date -> review, look, smell, taste, feel, overall

3NF Conversion-
Create relations r(ab) for all [a -> b]
- r1(beer, username, review_date, review, look, smell, taste, feel, overall)
Since the relation contains a candidate key, we do not need to create others.
There are no duplicates or redundant relations to remove.
r1 contains a candidate key for Reviews and is the only relation needed.