A Novel Approach towards Database Exploration using Collaborative Filtering

Ms. Kiran S. Sutar¹ Mr. Hemant A. Tirmare²

¹²Department of Technology Shivaji University, Kolhapur, India

Abstract—Recommender systems play an important role in today’s applications which need to satisfy diverse user needs. Recommendation systems are widely used in web applications, e.g. online book shop, movie recommendation etc. It also plays an important role in the field of database systems, where users retrieve interesting information from database by submitting SQL queries. In this case such a system suggests potentially useful queries to the user. In this paper we have presented a system which follows collaborative filtering technique. We have converted the queries submitted by user into general form which is suitable for further calculations. Our system maintains sessions of users. Session is nothing but sequence of queries submitted by the user. We first determines the past session which is similar to current user session, and subsequently use queries in that session to give recommendations.

Key words: SQL queries, SSD (Sky Server Database)

I. INTRODUCTION

Scientific and research communities, stores their huge amount of analytic data in the form of database. For example, SSD (Sky Server Database). In order to use this data to do some analysis or research work users generally submits SQL queries. Users interactively fires SQL queries to extract interesting information. But most of the users do not have expertise of using SQL. They can’t formulate complex SQL queries which contain aggregates and sub-queries. Even when users have complete knowledge of SQL they face trouble in finding useful information. This is because of such database systems are huge and contain large number of interrelated tables. Also, schema of the database is complex and may not be familiar to the users. That’s why they experience difficulties in discovering valuable information. To avoid such a problem and make database contents available easily to everyone, we require a system which will automatically suggests useful queries for the user depending on their interest. Such systems are called recommender systems.

We have proposed a system which will help users in formulating queries to explore the database. The proposed system will recommend queries which will extract potentially useful information from the database. Our system believes that if user A and B submitted same queries, then other queries of A may be useful for user B and vice versa. This means we are using queries of user A to help user B in his database exploration. The technique which we are using is nothing but Collaborative filtering which is very popular in web recommender systems. However we have adopted this technique for database context. Using collaborative filtering for query recommendation in the database context introduces several different challenges:

1) Syntactically Different Queries May Retrieve Same Information:
Consider following queries
Q1: SELECT P
Q2: SELECT P
FROM TBL1
WHERE Q=50;
Q2: SELECT TBL1.P
FROM TBL1 JOIN TBL2 ON (TBL1.R=TBL2.R)
WHERE TBL2.Q=50;
If tables TBL1 and TBL2 have a key/foreign key relationship on column R, then both queries retrieve same results. Such situations creates grate difficulties in the computation of similar queries and hence in the computation of similarities between users.

2) In the database context we don’t have any explicit rating system for the queries submitted by users. That’s why we cannot guess which queries are important for the computation of similarities between users.

3) Queries recommended by the system must be intuitive so that users can understand and edit it if necessary.

II. TYPES OF RECOMMENDATION APPROACHES

Recommender systems are very popular particularly in the web applications. There are various approaches taken by these systems.

A. Content-Based:

In content based systems each item is described by their features. For Instance, Let’s take an example of online book store. In this case book is our item and its features are category, author, publication etc. Similarity between different items is calculated by using these features. Past history of user is also taken into consideration. The items which are similar to the previously liked items by user are recommended to them. In this case past history of only current user is considered.

B. Collaborative Filtering:

These systems focus on determining user to user similarities, and recommends items liked by those similar users. Past history of other users is used to discover similar users. Such systems focus on user to user similarity. That’s why in [1] the author used the term “people-to-people correlation” for collaborative filtering. [2] Presents some modern extensions in the field of collaborative filtering technique. In [3] the author suggested item to item collaborative filtering technique for recommendation.

C. Demographic:

Demographic information can also be used for generating suggestions. The idea behind this approach is that similar suggestions should be generated for same demographic people. For example, the books suggested for the user depends on their language and qualification. Age also play important role in the suggestions. Even though such systems are quite popular in marketing literature, there has been
somewhat little proper research into demographic systems [4].

D. Community-Based:
Most of the people do what their friends advise them to do. Community based systems exploits such advices from user’s friends. The system follows the saying “Tell me who your friends are, and I will tell you who you are” [5] [6]. Evidence suggests that most of the people take advices from their friends than from similar but anonymous users [7]. This type of systems also called social recommender systems [8]. This type of recommender systems determines social relations of the user and subsequently uses the advices of user’s friends. These systems totally rely on the advices given by user’s friends.

E. Hybrid Recommender Systems:
Hybrid recommender system combines above mentioned approaches.

III. SYSTEM MODULES
Our system uses collaborative filtering technique for the generation of recommendations. In collaborative filtering, recommendations are generated based on first finding users who are similar to the current user. Subsequently the system recommends similar users queries. In interactive database exploration the user submits sequence of queries to the database system. Next query in the sequence may depend on the outcome of previously submitted query. We refer to this sequence of queries submitted by a user during one visit as a “session”.

A. Session Representation:
User first log-in the system. Then start exploring the database by submitting SQL queries one by one. Current user session is denoted by S0. Each query will get assigned a unique identifier, called query identifier QID. So the session is represented by sequence of queries.

B. Query Relaxation:
Query relaxation is the process of converting an SQL query into more general form. There are various algorithms to do this relaxation [9]. Our system uses simpler method to do this relaxation. The idea behind this relaxation process is to consider two queries similar if they query same table and columns using slightly different predicates. This makes easy to compute query similarity.

Query relaxation process mainly focuses on WHERE clause. All numerical and string predicates are replaced by general string representations. Table I shows the mapping of different types of predicates into general string representations. For example, all decimals are replaced by NUM, similarly set comparators are replaced by COMPARE[3].

<table>
<thead>
<tr>
<th>Predicate</th>
<th>General string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>NUM</td>
</tr>
<tr>
<td>String</td>
<td>STR</td>
</tr>
<tr>
<td>=,!=</td>
<td>EQU</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>COMPARE</td>
</tr>
</tbody>
</table>

Table 1: Query Relaxation table

Consider following query Q3.

Q3: SELECT actor_nm, cast

FROM movie
WHERE year=2000 AND awards>5;

If we relax above query Q3, we will get query Q4.

Q4: SELECT actor_nm, cast
FROM movie
WHERE year EQU NUM AND awards COMPARE NUM;

C. Query Fragmentation:
Once the query relaxation process is done next step is to fragment the query. It is the process in which we split out the query by separating its clauses. Each fragment will get assigned unique identifier, called fragment identifier (FID). SQL query is represented by sequence of FID’s. Fragmentation of above query Q4 will result in following fragments.

actor_nm, cast (f4)
Movie (f5)
Year EQU NUM AND awards COMPARE NUM (f6)

Each query is represented by sequence of FID’s. So query Q4 is now represented as follows:

Q4: {f4, f5, f6}

For each query our system generates query vector Q whose length is equal to number of fragments in the system. Q[0] is set to 1 if fragment Θ exist in query Q otherwise it is set to 0. So query vector for above query Q4 will be: {0, 0, 0, 1, 1, 1}

Consider following example,

Q5: SELECT branch_name
FROM branch;

Q6: SELECT branch_name, avg(balance)
FROM account
GROUP BY branch_name

D. Fragment Tagging:
If we split out above two queries we will get following fragments:

branch_name (f7)
branch (f8)
branch_name, avg(balance) (f9)
account (f10)
branch_name (f11)

In above example fragments having FID f7 and f11 are same but they have assigned different FIDs because they belong to different clauses. We have assigned tags to each fragment which indicate the clause to which that fragment belongs. Fragment tagging eliminates the ambiguities regarding the clause to which fragment belongs and thus helps to accurately calculate fragment similarity.

E. Select Clause Sequencer:
Consider following SELECT clauses “SELECT branch_nm, balance, branch_id”, “SELECT balance, branch_nm, branch_id”. These two clauses retrieve same columns but in different order. Our sequencer module sorts column names so that both Select clauses are considered as same. Select clause sequencer first separate outs column names in SELECT clause then sorts them in ascending order.

F. Session Representation:
Session is nothing but sequence of SQL queries submitted by user during one visit. Session for user i represented by a vector Si whose cell S[i] represents the important of
fragment $\emptyset$ in session $S_i$. Importance of fragment is calculated by counting number of occurrences of that fragment in that session. Suppose if there are 3 queries in session $S_i$.

$S_i = \sum_{q \in Q_i} S_q$

(1)

Above formula is used to represent session. This formula is adopted from [10]. Because of this way of session representation, we can easily find out fragments which occurred most frequently in that session. So session $S_i$ is represented as:

$S_i = \{0, 0, 1, 2, 2, 1, 1\}$

First two entries are zero because FID $f_1$ and $f_2$ does not exist in session $S_i$. FID $f_3$ occurred once so next entry is 1. Subsequently FID’s $f_4$, $f_5$ and $f_6$ are occurred two times in $S_i$, that’s why next three entries are all 2. In this way our system represents session. We have chosen this method to represent session because it is easy to calculate session similarities.

G. Similarity Calculation:

To calculate session to session similarity we have employed cosine similarity.

$cosine (A,B) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$

(2)

Using above formula we calculate session to session similarity and subsequently rank the sessions according to their similarity score. Here session of current user is denoted by A. Session of past users is denoted by B. Once most similar sessions are determined, our system retrieves the most frequently occurred queries in those sessions. Those queries are presented to user as recommendation. User can select any SQL query from those recommended queries. And also is able to edit any SQL query and then submit it.

IV. SYSTEM ARCHITECTURE

Figure 1 shows the architecture of our system. Users interact with our system using database query interface. User can browse the database schema, formulate queries and access database using this interface. Queries submitted by user are first checked for syntactical correctness. If user query is not syntactically correct it returns error message to the user. If it is correct then it forwards that query to the database and to user query profile. Database system will return result of query to the user. As described above query submitted by user is first relaxed and then fragmented to convert it into suitable format. Systems Query log contain all the queries recorded in the systems query logs. It uses cosine similarity for the calculation of similarity. Query ranking module uses these similarities to predict the “rank” (i.e. importance) of each session with regards to the current user session. The queries belonging to highest ranked sessions are taken into consideration. Now the queries belonging to these sessions are ranked according to their frequency, which means the queries which occurred most frequently are suggested first.

![System Architecture](image)

**Fig. 1: System Architecture**

V. CONCLUSION AND FUTURE WORK

In this paper we have represented framework of our SQL query recommendation system. Our system is divided into three main modules namely user query profile, systems query log and recommendation engine. We have maintained systems query log in the format suitable for the calculation of similarity. Our system follows collaborative filtering approach of recommendation: Future work for our system is support for sub queries.

REFERENCES


