XData: Clustering based on SQL query structural similarity

Team Members

1. Rohan Garg 160050018
2. Rohit Choudhary 160050030
3. Vikrant Garg 160050033
4. Sharvik Mital 160050059

Description

We have developed a module which will be helpful for grading done by current XData tool.

For each question, there can be multiple solutions (correct as well as incorrect) given by students, which may differ vastly in their implementation structure. For eg: using a join instead of a subquery.

In the present tool, for each different structure implementation, the instructor has to give a correct query. For this, the instructor has to go over all the students’ queries. We intend to make this a bit simpler for the instructor by clustering the student queries which have a similar structural implementation. The instructor can now directly give a correct query for each cluster.

Functional Specification

Given a set of queries, cluster them on the basis of structure ( join / from subquery / where subquery/group by / having/order by) and return the set of clusters.

Since we planned to add this feature in Xdata, we have made use of the QueryStructure class defined in the Xdata code. Apart from that, we have made clusters through our own implementation. The clustering service has been added to the XData package as a service. This service has been
employed by a servlet. After obtaining the assignment ID and question ID from the instructor, the servlet will obtain the query corpus from the database. After required processing is done and clusters are formed, queries for each cluster will be shown.

**Clustering of Queries**

**Approach 1**

Initially, we had planned on clustering the queries directly through Phase II (using KMeans[1]). For that, we used the same dimensions as used in Phase II and made the necessary modifications to an existing API for KMeans. Since KMeans requires a predetermined number of clusters, it was not possible to directly use it. We then thought of using Elbow method to dynamically find the optimum number of clusters. Unfortunately, the clusters formed did not take into account structural differences between the queries. This forced us to use two phases clustering.

**Approach 2**

This has been done in two phases for multi-level segregation.

**Phase I**

Queries are divided based on the higher level structural similarity which includes the type of set operator, the number of joins, the presence of subqueries in where and from clause, the presence of OrderBy and GroupBy.

For obtaining the sets, all queries are vectorized depending upon the attributes described earlier. Then, queries with the same vectors are grouped together to form a set. Later, sets will be passed onto Phase II clustering.

**Eg: assignment id=13, ques_id = 2, on running**

I. ```
   select student.ID, name
   from student, takes, course
   where student.ID = takes.ID and takes.course_id = course.course_id and
   course.dept_name = 'Comp. Sci.' and (year > 2010 or year<2010);
``` 

II. ```
    select student.ID, student.name
    from student, takes, course
    where student.ID = takes.ID and takes.course_id = course.course_id and
    course.dept_name = 'Comp. Sci.' and takes.year > 2010
    intersect
    select student.ID, student.name
```
from student,takes,course
where student.ID = takes.ID and takes.course_id = course.course_id and
course.dept_name = 'Comp. Sci.' and takes.year < 2010;

III. **WITH** studName(id,name,course_id,year,sec_id,semester) AS(
    SELECT id,name,course_id,year,sec_id,semester
    FROM(takes NATURAL JOIN student))
(SELECT id,name
FROM studName NATURAL JOIN course
WHERE year<2010 AND dept_name='Comp. Sci.')
INTERSECT
(SELECT id,name
FROM studName NATURAL JOIN course
WHERE year>2010 AND dept_name='Comp. Sci.');</p>

IV. select id,name
from (select *
    from ((select id
        from (select *
            from (select * from course natural join takes) as e
                where dept_name = 'Comp. Sci.' and year > 2010) as f)
        intersect
        (select id
        from (select *
            from (select * from course natural join takes) as e1
                where dept_name = 'Comp. Sci.' and year < 2010) as f1))
    as g natural join student) as h

We get three clusters and a set of 'rejected queries'(i.e. not handled by QueryStructure) after doing PHASE I clustering. Observe that queries I, II,III are indeed structurally different. Query I does not have \texttt{intersect}, Query II does not have \texttt{with} subquery(canonicalized to from subquery), Query III have subquery and \texttt{intersect} both.

Phase II
First, all queries within each set obtained from phase I are vectorized independently of other sets on dimensions based on selection conditions, join conditions, projected columns, GroupBy, having conditions, relation instances, subquery connectives, number of from subqueries and where subqueries. Since the number of clusters is not known, we apply the Adaptive Quality-based Clustering Algorithm on each set independently, implemented in Clustering Library[3].

Clustering is applied on only those sets which contain more than 3 queries.
Furthermore, we can add weights to each dimension while vectorizing the queries. Adding more weight to a dimension implies clustering will be finer on that dimension.

I. select distinct student.id, student.name 
   from student, takes, course, takes as four 
   where (student.id = takes.id) and (takes.course_id = course.course_id) and 
   (course.dept_name = 'Comp. Sci.' and ((takes.year < 2010 and four.year > 2010) 
   or (takes.year > 2010 and four.year < 2010)));

II. with std_ids(id) as (select id from takes as t, course as c where t.course_id 
   = c.course_id and dept_name = 'Comp. Sci.' and year != '2010')
   select distinct t1.id, name from std_ids as t1, student as t2 where t1.id = 
   t2.id;

III. select distinct student.ID, student.name from student, course, takes, takes as 
     takes2 where student.id = takes.id and takes.id = takes2.id and 
     takes.course_id = course.course_id and course.dept_name = 'Comp. Sci.' and 
     takes.year != 2010 and takes2.year != 2010 and ((takes2.year > 2010) and 
     (takes.year < 2010)) or ((takes.year < 2010) and (takes2.year > 2010));

Above are the queries that belong to the same set produced by phase I but belong to 
different clusters. Query I is different from Query II because later one uses with 
subquery. Query III is different from Query I because selection conditions are different in 
two queries.

**Output**

We provide two-way outputs.

1. After giving the assignment_id and question_id, the output is directly shown on 
   the window which shows clusters and queries within each cluster. Since there are 
two phases involved, the clusters formed after phase I is shown and also after 
phase II.

2. Also, there is a folder formed in the ‘/home’ directory which provides details about 
   the vectors formed and the clusters along with the ‘rejected queries’.
Challenges faced

1. As far as we could try, we were not able to find any previous work, which was related to clustering SQL queries on structural differences. So we had to come up our own idea.

2. While making the interface for entering question and assignment information to be sent to the clustering service, there was a problem with using HTTP requests because the service took too much time, and meanwhile, the request returned. So, we had to incorporate everything within the service itself (including the display of data).

Future Work Possible

1. As of now, the queries are processed by the QueryStruct class. The canonicalization done in it is minimal. To get more canonicalized queries, the clustering can be performed after using the Canonicalization performed by partialMarking. And while clustering, we can also check for the correct and incorrect queries.

2. For now, the clusters formed are checked by eyes. For better quantitative evaluation, a metric can be defined which provides how closely are the queries belonging to the same cluster and different to that of a different cluster related.

Bibliography

[2] Partial Marking for Automated Grading of SQL Queries