# PowerQ: An Interactive Keyword Search Engine for Aggregate Queries on Relational Databases

Zhong Zeng
National University of Singapore
zengzh@comp.nus.edu.sg

Mong Li Lee
National University of Singapore
leeml@comp.nus.edu.sg

Tok Wang Ling
National University of Singapore
lingtw@comp.nus.edu.sg

## ABSTRACT

Keyword search over relational databases has gained popularity due to its ease of use. Current research has focused on the efficient computation of results from multiple tuples, and largely ignores aggregate queries to retrieve statistical information from databases. The work in [6] developed a system that allows aggregate queries to be expressed using simple keywords. However, this system may return incorrect answers because it does not consider the semantics of objects and relationships in the database. In this paper, we present an interactive keyword search engine called PowerQ to answer aggregate queries. PowerQ extends the keyword query language and utilizes an ORM schema graph to capture the Object-Relationship-Attribute (ORA) semantics in the database. Given an aggregate query, PowerQ identifies the various interpretations of the query and applies aggregate functions and GROUPBY on the appropriate attributes of objects/relationships. Each query interpretation is denoted as an annotated query pattern, whose meaning can be described in natural language to facilitate user understanding. Through user interactions, PowerQ can determine the user's search intention, and translate the corresponding patterns into SQLs to compute the answers correctly. The PowerQ prototype is available at `http://powerq.comp.nus.edu.sg`.

## 1. INTRODUCTION

As databases increase in size and complexity, the ability for users to issue SQL queries has become a challenge. Keyword search over relational databases has gained popularity as it enables users to query the database without knowing the database schema or writing complicated SQL queries. Research on relational keyword search has focused on the efficient computation of results from multiple tuples [1, 2, 3, 5], and largely ignores queries involving aggregates and GROUPBY. The latter is also known as *aggregate queries*.

Aggregate queries provide a powerful mechanism to retrieve statistical information from the database. The work in [6] designed a prototype system called SQAK to handle aggregate queries. An aggregate query comprises of a set of terms and one of these terms is an aggregate function such as $COUNT$, $SUM$, etc. The terms in the query may match the names of relations or attributes or tuple values.

Consider the sample university database in Figure 1. Suppose we want to know the total credits obtained by the student Green, we can issue the aggregate query $Q_1$={Green SUM Credit}. However, we observe that incorrect answers may be returned by SQAK. For example, the term Green in $Q_1$ matches the names of two students $s2$ and $s3$ in Figure 1. This naturally implies that we should find the sum of the credits for each of these students, that is, the total credits for $s2$ is 5 while the total credits for $s3$ is 8. However, SQAK does not distinguish between these two "different" name matches, and outputs a total credits of 13 for students called Green, which is incorrect.

**Student**

| Sid | Sname | Age |
|---|---|---|
| s1 | George | 24 |
| s2 | Green | 18 |
| s3 | Green | 21 |

**Course**

| Code | Title | Credit |
|---|---|---|
| c1 | Java | 5.0 |
| c2 | Database | 4.0 |
| c3 | Multimedia | 3.0 |

**Enrol**

| Sid | Code | Grade |
|---|---|---|
| s1 | c1 | A |
| s1 | c2 | B |
| s1 | c3 | B |
| s2 | c1 | A |
| s3 | c1 | A |
| s3 | c3 | B |

**Teach**

| Code | Lid | Bid |
|---|---|---|
| c1 | l1 | b1 |
| c1 | l1 | b2 |
| c1 | l2 | b1 |
| c2 | l2 | b2 |
| c2 | l2 | b3 |
| c3 | l1 | b4 |

**Textbook**

| Bid | Tname | Price |
|---|---|---|
| b1 | Programming Language | 10 |
| b2 | Discrete Mathematics | 15 |
| b3 | Database Management | 12 |
| b4 | Multimedia Technologies | 20 |

**Lecturer**

| Lid | Lname | Did |
|---|---|---|
| l1 | George | d1 |
| l2 | Steven | d1 |

**Department**

| Did | Dname | Fid |
|---|---|---|
| d1 | CS | f1 |

**Faculty**

| Fid | Fname |
|---|---|
| f1 | Engineering |

**Figure 1: Sample university database**

Next, suppose we issue the query $Q_2$={Java SUM Price} to find the total price of the textbooks that are used in the Java course. The term Java matches a course title while the term Price matches an attribute of the $Textbook$ relation. This relation contains 3 foreign keys that reference the $Course$, $Lecturer$ and $Textbook$ relations respectively, and represents that a course can be taught by more than one lecturer using different textbooks. We see that there are 2 such textbooks, namely, $b1$ used by both lecturers $l1$ and $l2$, and $b2$ used by lecturer $l1$. But SQAK does not detect the duplicate textbook $b1$ by different lecturers of the Java course (i.e., $c1$) in the $Teach$ relation, and returns 35 for the total price. This answer is incorrect as students do not need 2 copies of a textbook for the same course.

In this work, we build a relational keyword search engine called PowerQ to answer aggregate queries correctly. PowerQ extends the keyword query language and utilizes the

ORM schema graph [7] to capture the Object-Relationship-Attribute (ORA) semantics in the database. Given an aggregate query, it identifies the various interpretations of the query and applies aggregate functions and GROUPBY on the appropriate attributes of objects/relationships. Each query interpretation is denoted as a graph called annotated query pattern, whose meaning is described in natural language. The query patterns that satisfy the user's search intention are translated into SQL statements to compute the answers. During the query processing, PowerQ utilizes the ORA semantics to distinguish the objects with the same attribute value and detect the duplications of objects/relationships regardless of whether the database is normalized or not. Otherwise, the aggregate function(s) cannot be computed correctly as we have shown in our example queries $Q_1$ and $Q_2$.

## 2. PRELIMINARIES

The work in [7] extends the keyword query language to include the keywords that match the names of relations and attributes. These metadata keywords provide the context of subsequent keywords and reduce the query ambiguity.

PowerQ further extends the query language to incorporate aggregates and GROUPBY. Thus, a keyword query $Q$ is a sequence of terms $\{t_1\ t_2\ \cdots\ t_n\}$ where each term $t_i$ either matches a relation name, an attribute name, a tuple value, GROUPBY or an aggregate function $COUNT$, $SUM$, $AVG$, $MIN$ or $MAX$.

### 2.1 ORM Schema Graph

The work in [8] classifies the relations in a database into object relations, relationship relations, mixed relations and component relations. An object (relationship resp.) relation captures the information of objects (relationships resp.), i.e., the single-valued attributes of an object class (relationship type). Multivalued attributes of an object class (relationship type) are stored in object/relationship component relations. A mixed relation contains information of both objects and relationships, which occurs when we have a many-to-one or one-to-one relationship. We call these semantics the Object-Relationship-Attribute (ORA) semantics.

The Object-Relationship-Mixed (ORM) schema graph is an undirected graph that captures the ORA semantics in the database. Each node in the graph comprises of an object/relationship/mixed relation and its component relations, and is associated with a type (object, relationship and mixed). Two nodes are connected if there exists a foreign key - key reference between the relations in these two nodes.

In Figure 1, the relations *Student*, *Course*, *Faculty* and *Textbook* are object relations while *Enrol* and *Teach* are relationship relations. Relations *Lecturer* and *Department* are mixed relations because of the many-to-one relationships between lecturers and departments, and the many-to-one relationships between departments and faculties respectively. Figure 2 shows the ORM schema graph of the database.

### 2.2 Query Patterns

Since keyword queries are inherently ambiguous, [7] introduces the notion of query patterns to represent the various interpretations of a query. These query patterns are generated from the ORM schema graph of the relational database.

Figure 3 shows one of the query patterns for the keyword query {Code George Green}. This patten depicts the

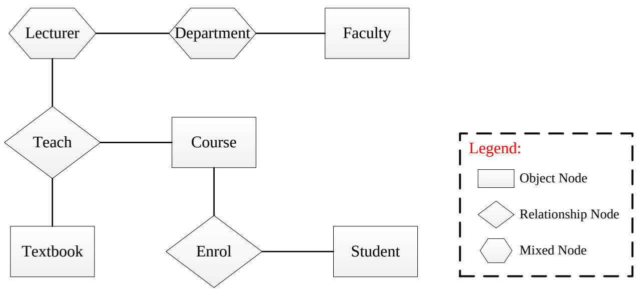


**Figure 2: ORM schema graph of Figure 1**

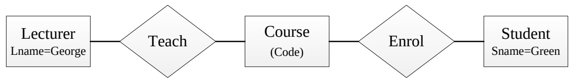


**Figure 3: Query pattern of {Code George Green}**

query interpretation to find information on the course which is taught by the lecturer George and enrolled by the student Green. To generate this query pattern, we identify the matches of each term in the query. The term Code matches the name of an attribute in the *Course* relation, while the terms George and Green match the values of the attribute *Lname* in the *Lecturer* relation and the attribute *Sname* in the *Student* relation respectively. Based on these matches, we know that Code refers to a course object, George refers to a lecturer object, and Green refers to a student object. From the ORM schema graph in Figure 2, the Course, Lecturer and Student nodes can be connected via a Teach and an Enrol node. Hence, we create these two nodes and obtain the query pattern in Figure 3.

PowerQ utilizes query patterns to capture the interpretations of an aggregate query. However, since an aggregate query includes aggregate functions and GROUPBY, we need to annotate the patterns to indicate the objects/relationships that aggregates and GROUPBY are applicable to. We will discuss how to achieve this in the next section.

## 3. SYSTEM ARCHITECTURE

PowerQ takes as input an aggregate query, and generates a set of SQL statements for the query patterns that satisfy the user's search intention. Figure 4 shows the architecture of PowerQ. The frontend of PowerQ interacts with the user during the query processing, while the backend communicates with the database and the ORM schema graph to compute the query answers. The main components in PowerQ are *Query Parser/Analyzer*, *Query Interpreter*, *SQL Generator*, *Visualization Module* and *Normalization Module*. The following sections give the details of these components.

### 3.1 Query Parser/Analyzer

Given an aggregate query, the Query Parser/Analyzer classifies the terms in the query into basic terms and operators. A basic term matches a relation name, or an attribute name, or a tuple value in the database, while an operator matches an aggregate function or GROUPBY. For the basic terms, the Query parser/Analyzer obtains their matches and determines the objects/relationships referred to by these terms based on the ORM schema graph of the database.

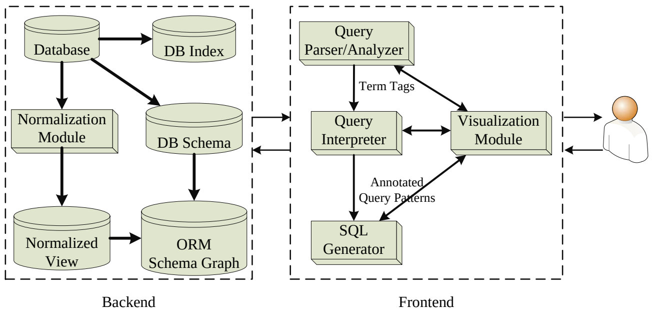


**Figure 4: System Architecture**

## 3.2 Query Interpreter

Next, the Query Interpreter generates a set of initial query patterns based on the basic terms of the query and the ORM schema graph of the database. Each query pattern contains a set of nodes that represents the objects/relationships referred to by the basic terms. Then, it annotates the query patterns with operators in the query. For each operator $t_i$, if its subsequent term $t_{i+1}$ refers to some object or relationship, then the Query Interpreter annotates the corresponding node with $t_i(id)$, where $id$ is the identifier of the object/relationship; otherwise, if $t_{i+1}$ refers to some attribute $a$ of an object or relationship, the Query Interpreter annotates the corresponding node with $t_i(a)$.

Consider the keyword query {COUNT Code George Green}. Figure 3 shows a query pattern obtained using the basic terms Code, George and Green. For the operator COUNT, since its subsequent term Code matches the name of an attribute in the *Course* relation, we will annotate the Course node with COUNT(Code), and obtain the annotated query pattern $P_1$ in Figure 5. This pattern depicts the query interpretation to find the total number of courses which are taught by lecturer George and enrolled by student Green.

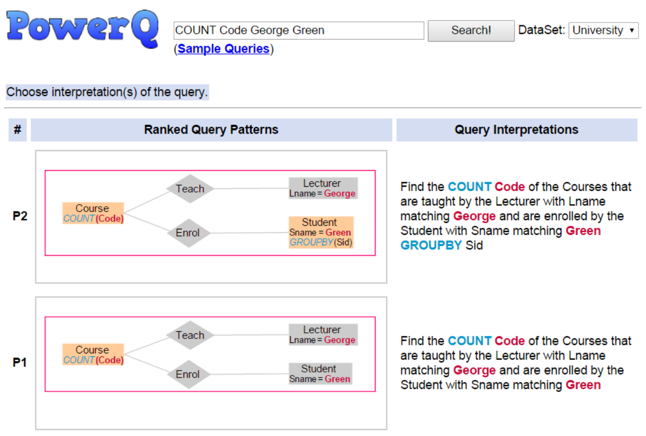
In an annotated query pattern, an object/mixed node with the condition $a = t$ refers to an object such that its value of attribute $a$ matches the basic term $t$. However, since this condition could be satisfied by more than one object in the database, we have two different query interpretations:

1. apply the aggregate functions(s) for every *distinct* object satisfying $a = t$; or
2. apply the aggregate function(s) for *all* the objects satisfying $a = t$.

The Query Interpreter distinguishes these two interpretations by annotating the object/mixed node in the pattern with GROUPBY($id$), where $id$ is the identifier of the object. By applying GROUPBY on object identifiers, we can distinguish objects with the same attribute value and compute the aggregate functions for each of them.

In Figure 5, the annotated query pattern $P_1$ contains a Student node that is annotated with the condition Sname = Green. From the database in Figure 1, we know that there are two students called Green. Hence, we have a second query pattern $P_2$ that is similar to $P_1$, except that we annotate the Student node in $P_2$ with GROUPBY(Sid). Figure 5 shows these two patterns: $P_1$ counts the number of courses for all the students called Green, while $P_2$ counts the number of courses for each student called Green separately.

Note that SQAK [6] does not distinguish $P_1$ and $P_2$, and thus may return incorrect answers to the query.



COUNT Code George Green

DataSet: University

Choose interpretation(s) of the query.

| # | Ranked Query Patterns | Query Interpretations |
|---|---|---|
| P2 | | Find the COUNT Code of the Courses that are taught by the Lecturer with Lname matching George and are enrolled by the Student with Sname matching Green GROUPBY Sid |
| P1 | | Find the COUNT Code of the Courses that are taught by the Lecturer with Lname matching George and are enrolled by the Student with Sname matching Green |

**Figure 5: Screenshot of annotated query patterns**

## 3.3 SQL Generator

The SQL Generator translates an annotated query pattern into an SQL statement to compute the answers. The straightforward approach is to join the relations of all the nodes, select the tuples that satisfy the conditions imposed by basic terms from the join result, and then apply aggregate(s) and GROUPBY on the selected tuples. However, this may generate an SQL that gives an incorrect answer.

Consider the query pattern $P_2$ in Figure 5. If we simply translate $P_2$ into an SQL that joins the relations *Course*, *Teach*, *Enrol*, *Lecterer* and *Student*, selects the tuples with conditions Lname = George and Sname = Green, and then applies the count aggregate and GROUPBY on the course code and the student id respectively, we will obtain wrong answers as the same course may be counted multiple times. This is because the Teach node in $P_2$ is in fact a ternary relationship involving course, lecturer and textbook objects (see the ORM schema graph in Figure 2). The same course can be taught by a lecturer using different textbooks. In other words, the same *Lid* and *Code* are duplicated for different *Bid* in the *Teach* relation.

To avoid this problem, PowerQ examines every relationship node $u$ in the pattern, and checks its corresponding node $v$ in the ORM schema graph. If the pattern only contains a subset of the participating objects in relationship $v$, then it projects the identifiers of these objects from $v$. This eliminates duplicates and PowerQ replaces the relation of $u$ with the relation obtained by this projection in the SQL.

For example, since the Teach node in $P_2$ only involves course and lecturer objects, PowerQ generates a subquery "SELECT DISTINCT Lid, Code FROM Teach" to project the attributes *Lid* and *Code* in the *Teach* relation. This subquery has a "DISTINCT" keyword, thus eliminating duplicates of $\langle Lid, Code\rangle$. We use this subquery result to join the other relations in the FROM clause as follows:

```
SELECT S.Sid, COUNT(C.Code)
FROM Lecturer L, Course C, Enrol E, Student S
     (SELECT DISTINCT Lid, Code FROM Teach) T
WHERE L.Lid=T.Lid AND C.Code=T.Code AND
      S.Sid=E.Sid AND C.Code=E.Code
GROUP BY S.Sid
```

Note that SQAK does not detect the duplicates of courses in Teach relationships, and thus returns incorrect answers.

## 3.4 Visualization Module

A keyword query is inherently ambiguous. However, the user who issues the query often has some particular search intention in mind [4]. The Visualization Module represents the various interpretations of a keyword query, and actively interacts with the user to obtain the interpretations that satisfy the user's search intention. In particular, if a term has multiple matches in the database and refers to different objects/relationships, the user is offered the opportunity to choose the matches. Further, if more than one query pattern is constructed for the query, the user is again allowed to choose his/her intended query patterns.

One feature of PowerQ is that it represents query interpretations visually and describes them in human natural language in order to facilitate users' understanding. For instance, the annotated query pattern $P_2$ in Figure 5 is represented as a graph annotated with the ORA semantics. The nodes with operators in the graph are highlighted to indicate the objects/relationships that aggregates are applicable to. The description of this pattern is to "Find the count of the courses that are taught by the lecturer with name matching George and are enrolled by the student with name matching Green group by Sid". The user can easily identify the intended query interpretation by the graph structure, and verify its meaning by the description. After the user chooses a query pattern, PowerQ computes the answers and represents them according to the corresponding search intention. Figure 6 shows the screenshot of the interface which displays the query answers for the query pattern $P_2$ in Figure 5 and the detailed information for user to verify the answers.

## 3.5 Normalization Module

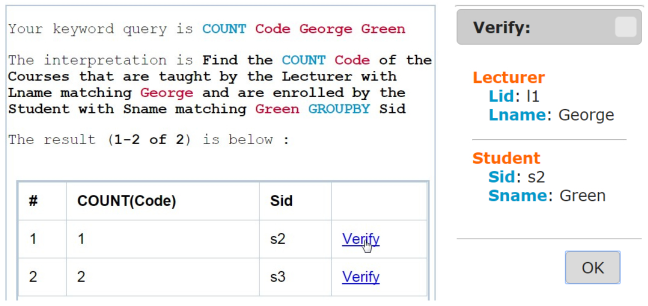
Relations in a relational database are often denormalized to improve query processing performance. This denormalization process will duplicate information of objects and relationships in the database and SQAK may obtain incorrect answers for an aggregate query.

PowerQ is able to detect denormalization and keep track of the object/relationship information in the database to answer aggregate queries correctly. This is achieved by examining the functional dependencies hold on the relations. If the database is denormalized, then it generates a normalized view of the database which comprises of a minimal set of normalized relations, and obtains the mappings of relations in the normalized view and the original schema. The normalized view is used to construct the ORM schema graph of the denormalized database and build query patterns of the query, while the mappings are used to generate the SQL statements which continue to compute the answers correctly.

## 4. DEMONSTRATION

In this demonstration, we will present a web-based browser interface of PowerQ, which communicates with the Java based server. The system is available at `http://powerq.comp.nus.edu.sg`. We intend to show the use of PowerQ against a number of real application scenarios such as the ACM Digital Library (`dl.acm.org`), and the IMDB database (`www.imdb.com`).

The demonstration will include three parts. First, we will run a number of sample aggregate queries against these resources. We will demonstrate how PowerQ exploits the ORA semantics in the database, distinguishes objects with the same attribute value, and detects duplications of objects in



```
Your keyword query is COUNT Code George Green

The interpretation is Find the COUNT Code of the
Courses that are taught by the Lecturer with
Lname matching George and are enrolled by the
Student with Sname matching Green GROUPBY Sid

The result (1-2 of 2) is below :
```

| # | COUNT(Code) | Sid | |
|---|---|---|---|
| 1 | 1 | s2 | Verify |
| 2 | 2 | s3 | Verify |

Verify:

Lecturer
Lid: l1
Lname: George

Student
Sid: s2
Sname: Green

**Figure 6: Screenshot of answers to query pattern $P_2$**

relationships to answer aggregate queries correctly. The user can run queries without aggregate functions or GROUPBY to verify the answers of the aggregate queries. Next, we will run the aggregate queries on the denormalized data. We will demonstrate how PowerQ continues to process the aggregate queries correctly. Finally, the user will be free to run their own queries.

Through this demonstration, we will highlight the importance of the ORA semantics to relational keyword search. This is reflected in three aspects. First, the interpretation of keyword queries requires the system to be knowledgeable about the ORA semantics. Second, in order to answer queries involving aggregates and GROUPBY correctly, we need to distinguish objects with the same attribute value and detect duplications of objects in relationships based on the ORA semantics. Third, we need to keep track of the ORA semantics in the database, so that queries on denormalized databases can continue to be handled correctly.

## 5. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *VLDB*, 2002.

[2] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, and Y. Velegrakis. QUEST: A keyword search system for relational data based on semantic and machine learning techniques. *Proc. VLDB Endow.*, 2013.

[3] M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu. MeanKS: Meaningful keyword search in relational databases with complex schema. In *SIGMOD*, 2014.

[4] F. Li and H. V. Jagadish. Usability, databases, and HCI. *IEEE Data Eng. Bull.*, 35(3):37–45, 2012.

[5] Y. Luo, W. Wang, and X. Lin. SPARK: A keyword search engine on relational databases. In *ICDE*, 2008.

[6] S. Tata and G. M. Lohman. SQAK: Doing more with keywords. In *SIGMOD*, 2008.

[7] Z. Zeng, Z. Bao, T. N. Le, M. L. Lee, and T. W. Ling. ExpressQ: Identifying keyword context and search target in relational keyword queries. In *CIKM*, 2014.

[8] Z. Zeng, Z. Bao, M. L. Lee, and T. W. Ling. A semantic approach to keyword search over relational databases. In *ER*, 2013.