Faculty of Computer Science
Institute of Systems Architecture, Chair for Computer Networks

Diploma Thesis

Calculating Similarity of Arbitrary Reports

Veronika Thost

Professor: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Tutor: Dr.-Ing. Daniel Schuster (TU Dresden)
External Tutor: Dr.-Ing. Konrad Voigt (SAP Research Dresden)

Submitted August 31, 2012

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Systemarchitektur, Professur Rechnernetze

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

Name, Vorname:	Thost, Veronika	
Studiengang:	Informatik	Matr.-Nr.: 3399246
Thema:	**Calculating similarity of arbitrary reports**	

ZIELSTELLUNG

One fundamental concept of business intelligence (BI) is the report. A report is a visualization of a query applied on certain data sets. Reports are traditionally used by business experts but created by IT experts. An ongoing transformation is the shift in report creation to enable business experts to create their own reports on their own local data. The local knowledge can be shared by adapting one local report of one expert to the data of another.

While adapting reports several challenges occur: Different data sets used for the reports, different visualizations, and different tools for report creation. To tackle these challenges an intermediary model and combination system is needed. Such a system is developed in the Remix project at SAP Research providing recommendations and visualizations in the steps of adapting data sources, queries and aggregation functions and the final visualization. These recommendations can be provided by semi-automatic matching techniques. While the matching data and metadata is a well explored field, the matching of reports still remains a challenge.

SCHWERPUNKTE

- Foundations: Related work on query models (e.g. SQL), query similarity, and feature-based similarity to support report matching
- Concepts for report similarity calculation of the query layer, data layer, and optionally visualization layer using schema matching and feature-based similarity
- Evaluation using a suitable reference data set (either defined or extracted) including different reports and manual mappings between the reports
- Assessment and comparison of the overall quality of the report similarity calculation

Betreuer:	Dr.-Ing. Daniel Schuster
Externer Betreuer:	Dr.-Ing. Konrad Voigt (SAP Research)
Betreuender Hochschullehrer:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Beginn am:	01.03.2012
Einzureichen am:	31.08.2012

Unterschrift des betreuenden Hochschullehrers

Abstract Reporting is an essential part of business, today, and people spend a lot of time creating meaningful visualizations of their most important data. Surprisingly, the reuse of reports (i.e., applying the same visualization or query on different data) is not common. The recommendation of proven, existing queries represents one part of this reuse. Since there are several report formats and the reports target different data sources, the task of matching report queries for recommendation is very complex and has not been addressed, yet. Recent works that aim at query recommendation – usually to support the user in database add-ons – focus on collaborative approaches or target query completion. The idea behind this study is to follow a content-based, combined approach to match report queries. The use of both an abstract and a concrete representation of the queries allows the application of different, well-known matching techniques in parallel. The focus of this work is to evaluate the impact of similarity search and schema matching for matching queries. For that, query matching algorithms based on the two techniques are developed: an efficient, index-based comparison using similarity search and a fine-grained matching of the parse trees of the queries with schema matchers. Next to the presentation of an effective combination of those algorithms, a major contribution of this thesis is the creation of an empirical data set of 150 queries with the corresponding similarity ratings for all 22,500 query pairs for a comprehensive evaluation.

Contents

1. Introduction

Reporting is an essential part of business, today, and people spend a lot of time creating good looking and meaningful visualizations of their most important data. Surprisingly, the reuse of reports (i.e., applying the same visualization or query on different data) is not common. Therefore, SAP Research addresses this problem with the development of remix [58], a tool that supports the user in report creation. A report presents the visualization of a query on a data source (e.g., a bar chart presenting the average salary of employees recorded in a database). And remix assists the user by proposing, for example, a specific visualization for a data source, which has been mined from a similar, existing report.

This thesis supports the finding of similar reports by calculating similarities for arbitrary reports. It focuses on matching the queries contained in the reports and proposes an approach for *content-based* query recommendation. Thus, it helps to overcome one major challenge of report creation, query formulation.

State-of-the-art reporting tools usually provide auto completion regarding table and column names to assist the user with query formulation [3, 6]. Other approaches to support the user include recommender systems [22, 38, 78], visual query languages [5, 21], keyword search in databases [57, 67], and the use of predefined example queries [8, 14]. However, the existing tools do not focus on the contents of the queries (e.g., there is no automated support for a user who has a complex query in mind but needs help with the formulation). Hence, system-independent recommendation of proven, similar examples (i.e., queries from arbitrary real-world reports) is not provided by them – in particular, because of the difficulties with *query matching*, which compares the query issued by a user to the queries stored in the system.

1.1. Motivation for Query Matching

The task of matching two queries is non-trivial. Especially, if it targets a precise comparison. Real-world queries occur in endless numbers, different formats, and refer to diverse environments. In addition, semantic differences and inconsistencies between queries are objections that need to be considered. There are, further, different design possibilities to describe semantically equivalent concepts. Although standardized query languages exist, their multiplicity and plenty implementations create a very heterogeneous environment for a matching system, or *matcher*, to act in. Moreover, the data a query describes may be very domain-specific, which is reflected in the language used, too. It is not possible to cope with these syntactic and semantic differences without a fine-grained and comprehensive consideration of queries and sound reasoning.

The challenges outlined above hint at different aspects of and information about queries. Next to the textual consideration of the query (i.e., a purely syntactic consideration), its semantics (i.e., how the syntax is interpreted), and external information (e.g., about the environment) can be exploited for calculating similarity of queries.

The motivation behind this work is to explore *content-based* query matching techniques (i.e., approaches that focus on the information contained in the queries) that could be used to calculate the similarity between queries. Based on this analysis, a combined query matching approach is to be developed. The problems mentioned above imply the need for comprehensive matching techniques, which apply the information present in queries to determine their similarity. Therefore, it makes sense to combine existing methods that are able to capture individual aspects of queries to calculate a similarity that reflects several dimensions of existing information.

Query matching with the goal of recommending queries to a user is a very recent area of research, which is conceptually described by Khoussainova et al. [50]. Implementations are provided by few advanced query completion approaches [47, 51, 87] and recommender systems [16, 22, 37, 38, 48, 77, 78]. The latter, however, focus more on user preferences than the queries themselves to determine query similarity. Earlier studies that include query matching aim at supporting the users of database systems [23, 34] or target query optimization [86, 89, 91]. But, none of the existing systems exploits the information given with the queries to a maximum (e.g., by using a combination of several, useful query matching approaches). In addition, there exists neither a general classification of query matching techniques nor appropriate evaluations that measure the matching quality. However, a thorough exploration and evaluation of query matching approaches would permit the creation of a query matching algorithm that makes the most of the information it is given.

1.2. Scope of this Thesis

This project aims at creating a query recommendation system for remix, a Business Intelligence (BI) tool currently developed by SAP Research. This query recommendation system recommends queries using query matching – by calculating similarity values for queries based on a comprehensive comparison. To the best of our knowledge, such a matcher does not exist, to date. The main difference to other query matching approaches is that it considers various aspects of the input (i.e., it combines different matching algorithms) and delivers a single value as an overall similarity. Furthermore, its quality is assessed in an extensive evaluation. In particular, the performance of traditional schema matching algorithms (e.g., name matching) is examined. Marcel and Negre [61] state that, to date, there are no adequate data sets to evaluate query recommendation systems (i.e., sets of real-world queries where the similarity for pairs of queries has been rated by real persons). Hence, the construction of such a reference data set presents a major contribution of this thesis.

The structure of this work is as follows: After a short introduction of the terminology (in Chapter 2), a detailed analysis of the query matching problem describes the task to be solved (in Chapter 3). Related work in query recommendation and matching is presented and classified in Chapter 4. In the main part, a combined query matching approach integrating several content-based matching methods is conceptualized (in Chapter 5). Finally, a reference data set is developed, and the query recommendation system is evaluated in an empirical evaluation (in Chapter 6).

2. Preliminaries

This chapter introduces the theoretical foundations of this work. First, it describes the concept of queries. Then, it covers preliminaries in the fields of similarity search and schema matching, which are the methods the combined query matching approach proposed in this thesis builds upon. Lastly, selected measurements for evaluating query recommendation and matching are presented.

2.1. Queries

The focus of this work is on matching queries extracted from reports. Such a query is a unit of text formulated in a specific query language. The latter usually depends on the tool that was used to create the report. Query languages generally refer to a particular data model the sources queried have to correspond to. These data models can be classified according to the degree of structure they maintain (i.e., unstructured, semi-structured, or structured) and, furthermore, according to the type of structure they create (e.g., a tree, a relational, or a multidimensional structure). This work focuses on queries over relational data sources since they are the models most common for current data warehouses. The most popular relational query language is SQL [10].

In order to reason about queries, a general data model describing the results of the query is insufficient. This is because queries describe complex operations on data, which need to be included in a detailed analysis (i.e., the semantics of the language need to be considered, too). For that reason, theoretic work on queries usually uses relational algebra to express relational queries. Also, several extensions of relational algebra have been developed to capture the features provided by current relational query languages, in the past. Exemplary translations of two SQL queries into relational algebra are given in Example 2.1.

Example 2.1. Below, two SQL queries, q_0 and q_1, are shown together with the respective representations in relational algebra.

---

q_0:

```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

$\pi_{name,sales}(\sigma_{manager='Chang'}(department))$

q_1:

```
SELECT name
FROM dept d JOIN manager m ON d.id=m.deptid
WHERE totalsales >50000
```

$\pi_{name}(\sigma_{totalsales>50000}(\rho_{d/dept}(dept) \bowtie_{d.id=m.deptid} \rho_{m/manager}(manager)))$

2.2. Similarity Search

This section gives an overview of similarity search, a methodology for comparing objects. In this thesis, similarity search is applied for a textual comparison of queries. Therefore, the concept of index search and a function to capture the degree of similarity between two textual documents, which is used in the search engine library Apache Lucene [1], are described, in the following.

Similarity search addresses the *nearest neighbor problem*, an optimization problem for finding closest points in, most often, metric spaces. On the basis of Skiena [82], the problem is described as follows:

"Given a set P of points in a metric space of n dimensions and a point q, find a point in P that is closest to q."

Similarity search is used to compare a given object against a collection of objects in order to find those that are most similar to the given object. In case a metric space is considered, the objects can be seen as points in the space and are defined by vectors in its dimensions, as it is described by Zezula et al. [90]. Hence, the problem of matching the objects is reduced to a vector comparison. The similarity of the objects is then defined by the distance of the vectors. That distance is usually computed by means of the Euclidean Distance[1] or the Manhattan Distance[2].

2.2.1. Index Search

Similarity search as a textual comparison can be based on an inverted index. An inverted index is a data structure that supports fast full text search for a set of documents. For that, the contents of the documents to be stored are processed and stored separately (e.g., individual words), in so-called *fields*, with references to the documents they are part of. As a consequence, a search for specific content can be performed efficiently and immediately yields the corresponding documents. Typical algorithms achieve a logarithmic complexity in dependence of the number of unique terms in the documents [45, 64].

2.2.2. Calculating Similarity Values for Text

The calculation of query similarity regarding specific terms in the queries is based on Information Retrieval (IR) methods that search specific terms in a set of documents. In particular, it uses a combination of the Boolean model of Information Retrieval[3] and the Vector Space Model of Information Retrieval[4], which is described in the Apache Lucene documentation [12]. Besides the combination of the two approaches, Apache Lucene includes further factors

[1] The Euclidean Distance represents the general straight-line distance between two points in an n-dimensional Euclidean space. Given two points p and q, it is defined as $d_{euclid}(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$ [85].

[2] The Manhattan Distance describes the length of the shortest paths between two points p and q that go along horizontal and vertical segments. It is defined regarding the Euclidean plane (i.e., the two-dimensional Euclidean space). For the points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, it is calculated as $d_{manhattan}(p,q) = |x_p - x_q| + |y_p - y_q|$ [18].

[3] The Boolean model of Information Retrieval considers the documents to be searched as sets of terms. The search query consists of terms and operators from boolean logic, and documents are retrieved if they fulfill that search query [84].

[4] In the Vector Space Model of Information Retrieval, the search query and the documents are seen as vectors in an n-dimensional space for n separate terms. The relevance of a document w.r.t. the query is then computed based on the values in the vectors [75].

in the calculation. The formula specified for the Apache Lucene Project [1] may be simplified, because this work omits the boosting of specific terms or documents (i.e., queries). This is because the boosting of specific information would make assumptions about the input (i.e., what parts are more important), which cannot be made regarding an arbitrary set of queries, in general. Let q_s be a search query looking for specific terms in a set of documents, in the following. Accordingly adapted, the function calculating a score for a document d w.r.t. q_s is described below together with its individual factors:

$$fscore(q_s, d) = coord(q_s, d) * qNorm(q_s) * \sum_{t \in q_s} (tf(t, d) * idf(t)^2)$$

- $tf(t, d)$, representing the term frequency, describes how often a term t appears in document d (i.e., $tf(t, d) = \sqrt{frequency}$).
- $idf(t)$, the inverse document frequency, is a factor that weighs higher terms that only occur in few documents (i.e., given the set D of all documents, $idf(t) = 1 + \ln \frac{|D|}{|\{d|d \in D, t \in d\}|+1}$).
- $coord(q_s, d)$ includes the total similarity of the terms in both the query and the document under consideration by specifying how many of the terms of the query are found in the document.
- $qNorm(q_s)$ aims at normalizing the similarity value of one search such that different search tasks become better comparable. It is the inverse of the square root of the sum of the squared inverse document frequencies (i.e., $qNorm(q_s) = \frac{1}{\sqrt{\sum_{t \in q_s} (idf(t))^2}}$).

Note that the above formula, especially, supports documents with contents referring to different kinds of fields, which are comparable to aspects (e.g., considering a book as document, its terms could be discerned according to their occurrence in the title, text, or ISBN). Then the term frequency is evaluated w.r.t. the respective fields instead of the whole document.

2.3. Schema Matching

In this thesis, special attention is given to schema matching algorithms. They are applied in the concept of this work for a detailed syntactic comparison of queries. Since the parse tree of queries can be regarded as a schema structure, schema matchers naturally can be applied for matching them. This section presents foundations of schema matching, selected schema matching algorithms, and approaches to combine the results of several matching algorithms. The latter represent an important means to form complex, combined matching algorithms.

A *schema* basically is a tree. In the following, its nodes are called *elements* and have a name associated with them. The leaves of the tree correspond to the attributes in the schema. Attribute nodes are additionally annotated with type information. Based on Rahm and Bernstein [70], the schema matching problem is described as follows:

"Given two schemas as input, produce a mapping between the elements of the two schemas that correspond semantically to each other."

Schema matching algorithms, so-called *schema matchers*, map the elements of two schemas that correspond semantically to each other. For that, the cartesian product of element-to-element similarity values is computed between the schemas. It constitutes a matrix of pair-wise

similarity values. The values in this matrix determine the degree of correspondence between the pairs of elements. Elements that correspond to each other are called *matches* for each other.

Schema matching algorithms apply linguistic methods, structural analysis, domain knowledge, and past mappings in order to create a mapping between two database schemas. Several traditional schema matchers, evaluated in this study, are described next.

2.3.1. Schema Matchers

Rahm and Bernstein [70] give a good overview of different kinds of schema matchers. This thesis considers only matchers that work *schema-only* based (i.e., they do not consider information contained in instances, which do not exist in the query context). Nevertheless, *element-level* (i.e., for matching two elements only the information in those two elements is considered) as well as *structure-level* matchers (i.e., also information of other, usually related, elements is considered for matching a pair of elements) are evaluated regarding query matching. Most matchers concentrate on the names and text contained in the schemas and are thus *linguistic* approaches. On the contrary, *constraint-based* approaches focus on other information that would restrict the possible instances of the schema (e.g., type information). The different traditional matchers that are considered in this thesis are described below.

Name Matcher The Name Matcher compares the names associated with the schema elements. According to Rahm and Bernstein, *"Name-based matching matches schema elements with equal or similar names."* For the computation of a concrete similarity value, established string similarity algorithms [25] can be applied.

Path Matcher The Path Matcher is an extension of the Name Matcher. It considers not only the names of the elements to be matched, but also those of the respective ancestors in the schema. While retaining the order of the ancestors, the Path Matcher combines their names to a so-called *name-path*. Then, two elements are matched by comparing the name-paths associated with them.

Type Matcher The Type Matcher performs a comparison of the data types of all attributes in the schemas. For that, it applies a predefined mapping, which specifies the similarity between different data types.

Parent Matcher The Parent Matcher compares two elements w.r.t. their direct ancestors, or *parents*, in the schema. Thus, it assumes two elements to be similar if their parents correspond to each other. Note that, to determine the correspondence of the parents, their similarity has to be known (e.g., it may be computed in advance with another matcher).

Children Matcher Analogous to the Parent Matcher, the Children Matcher matches a pair of elements by comparing the respective sets of direct successors in the two schemas. Thereby, it has to apply a function that determines the similarity of two sets of schema elements.

Sibling Matcher The Sibling Matcher compares two schema elements by using information about their siblings (e.g., if there is an overlap in the sets of the siblings of two elements). Its processing, thus, is similar to that of the children matcher.

Leaf Matcher The Leaf Matcher matches two elements by comparing all those of their successors that are leaves (i.e., attributes in the schema). Hence, it classifies two schema elements as similar if the attributes finally descending from them are similar.

Since schema matchers perform a pair-wise comparison between all elements of two schemas, the complexity of the algorithms is at least quadratic in the number of elements in the schemas. However, some of the matchers described above, namely the Children and the Sibling Matcher, even have a cubic worst-case complexity. The latter can be overcome in the implementation by buffering the pair-wise similarity values between all the elements in the schemas matched.

2.3.2. Similarity Values of Schema Matchers

Schema matching systems like COMA [29] and Cupid [59] often apply several matchers in a row in order to achieve better results. Further, the individual matchers compute so-called *similarity values* for all element pairs. This gives rise to the question how the similarity between two schema elements is to be determined by *aggregating* the results of several matchers, on the one hand. On the other hand, strategies for *selecting* the similarity values that really express a correspondency (e.g., all values above a given threshold) have to be developed. Strategies for both have been proposed in the last decade, an overview is provided by Peukert et al. [69]. In particular, this topic is addressed by Do and Rahm [28, 29], who specify several methods that are described in the following paragraphs.

Do and Rahm discern four aggregation strategies: Max, Min, Weighted, and Average. As it is indicated by the names, Max uses the maximum of the given values and Min the minimum, while with both strategies other values are disregarded entirely. In contrast, Weighted considers the values of all matchers by weighting them according to a predefined scheme (e.g., such weights can be found using machine learning). Average presents a specific case of Weighted where all matchers are assigned the same weight.

The strategies developed by Do and Rahm for selection are MaxN, MaxDelta, and Threshold. A schema matching algorithm calculates similarity values for all element pairs between two schemas. A selection strategy can be described as a function applied to the result of such a matcher. The two strategies MaxN and MaxDelta, in particular, consider the similarity values in the context of a specific match task.

For the description of the selection strategies, consider two ordered sets S and T containing all the elements of two schemas that are matched, respectively. The matrix $M_{S,T}$ contains the results of matching S and T, the similarity values $v \in [0..1]$, calculated for all pairs of elements between the two schemas. A selection strategy adapts the values in this matrix. $M_{S,T}$ is indexed with the positions of the elements in the sets (i.e., m_{ij} denotes the similarity value between the elements at positions i and j in S and T, respectively). Further, $n\_rowmin(n, s, M_{S,T})$, $n\_rowmin : [0..|T|] \times [0..|S|] \times M_{S,T} \rightarrow [0..1]$, extracts the n best matches for a schema element at index s in S w.r.t. a specific match task and returns the minimal similarity value regarding the elements of that set (i.e., the minimum in the *sth*-row of $M_{S,T}$). Function $n\_colmin : [0..|S|] \times [0..|T|] \times M_{S,T} \rightarrow [0..1]$ proceeds equivalently w.r.t. the elements in T. The selection strategies, next, are described as functions adapting the values $m_{i,j}$ in $M_{S,T}$, the result given by a matcher.

MaxN considers only those matches similar with similarity values at least equal to the minimum of the n best values for a given parameter n:

$$select_{MaxN}(n, m_{i,j}) = \begin{cases} m_{i,j} & \text{if } m_{i,j} \geq min(n\_rowmin(n, i, M_{S,T}), n\_colmin(n, j, M_{S,T})) \\ 0 & \text{else} \end{cases}$$

Above, *min* denotes a function selecting the minimum of two values. MaxDelta considers those matches similar having values within the range of a certain interval, which is derived from the maximum value of the best match using a special delta-parameter $d \in [0..1]$:

$$select_{MaxDelta}(d, m_{i,j}) = \begin{cases} m_{i,j} & \text{if } m_{i,j} \geq (1-d) * min(n\_rowmin(1, i, M_{S,T}), n\_colmin(1, j, M_{S,T})) \\ 0 & \text{else} \end{cases}$$

Lastly, Threshold does select all values that are at least as large as a predefined threshold $t \in [0..1]$:

$$select_{Threshold}(t, m_{i,j}) = \begin{cases} m_{i,j} & \text{if } m_{i,j} \geq t \\ 0 & \text{else} \end{cases}$$

A detailed example for the application of aggregation and selection strategies is given in Chapter 5.

2.4. Measurements

In the area of recommender systems, the recommendation quality is usually measured regarding the three properties overall usage (i.e., if the recommendations presented by a system are helpful altogether), rating (i.e., if the importance of a recommendation is predicted correctly), and ranking accuracy (i.e., if the order of the recommendations makes sense). This section introduces the measures applied in this study. Since the focus of this work is on matching queries for finding recommendations, the usage accuracy, presented first, is considered most important. Measures from the other classes are only covered marginally.

2.4.1. Usage Accuracy

To measure the usage accuracy, traditional metrics from Information Retrieval can be considered – an overview is presented by Singhal [81]. Three of the probably most important measures, *precision*, *recall*, and *F-measure*, are described in this section. Further, *precision@k*, an adaptation of precision for the recommendation case is regarded.

In Information Retrieval, the quality of recommendation is usually measured by comparing the recommendations provided by a system to those that are classified as recommendations in a reference set. For an example, consider Figure 2.1. It shows two ovals: the grey-marked recommendations R returned by a system, the so-called *positives*, and the recommendations of a reference set R'. Both R and R' are sets of pairs containing a recommendation and a corresponding rating value. According to Do [28], there are three subsets, which describe the recommendations that were identified correctly, I (i.e., $I = R \cap R'$), those that were falsely classified as recommendations, F (i.e., $F = R \setminus I$), and the recommendations that were missed, M (i.e., $M = R' \setminus I$).

Based on this classification, there are a number of quality measurements, presented in the following.

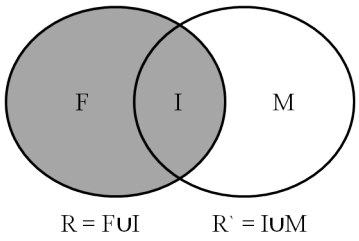

Figure 2.1.: The recommendations R of a recommendation system compared to the recommendations R' of a reference set

Precision describes the accuracy of the recommendation by relating the recommendations identified correctly with all recommendations determined. It is defined as

$$pr = \frac{|I|}{|R|} = \frac{|I|}{|I|+|F|}$$

Recall describes the completeness of the result. It is represented by the share of real recommendation that were found, and specified as

$$re = \frac{|I|}{|R'|} = \frac{|I|}{|I|+|M|}$$

F-Measure combines precision and recall in order to neglect none of them in favor of the other (e.g., precision could easily be maximized by returning only very few correct recommendations, but disregarding the retrieval of a complete result). It is defined as the harmonic mean of precision and recall, per default[5]:

$$F_1 = \frac{2 * pr * re}{pr + re}$$

Note that precision, recall, and F-measure, all range between 0 and 1, with 1 representing the best value. The three metrics are usually applied to measure the quality of matching algorithms [28, 29]. Such algorithms compare two sets of items to each other to find all items that are similar.

In this work, however, the goal is finding the best recommendations. Thus, the evaluation should focus on the (sub)list of actual recommendations of a system. For that, the set-theoretic approach of the three measures presented above is often not adequate. Especially, the information given by recall, in this context, is not very meaningful. Hence, in this work, the query matching approaches are primarily evaluated regarding the k topmost results, which would present their recommendations. The corresponding measure is called *precision@k*.

Precision@k describes the accuracy of the result regarding a given cutoff rank k. Considering the set $R_f \subseteq R$ of the best k recommendations in R (i.e., $|R_f| = k$), it is defined as

$$pr@k = \frac{|I \cap R_f|}{|R_f|}$$

[5]Note that F-measure, in general, is a metric relating precision and recall. Its definition depends on a variable β, which specifies the weighting of the factors. It is calculated as $F_\beta = (1 + \beta^2) * \frac{pr*re}{\beta^2*pr+re}$. The default version calculating the harmonic mean uses $\beta = 1$ and is also called F_1-Measure [20].

The measures presented next are more specific for the recommendation case. Although it is not explicitly stated, they are evaluated w.r.t. a given cutoff rank k. Hence, they are defined regarding the set of topmost recommendations $R_f \subseteq R$.

2.4.2. Rating Accuracy

To determine the rating accuracy of a system, the rating values predicted by the system have to be compared to those given in a reference set. Corresponding metrics usually measure the error of the system. This work considers the Mean Absolute Error (MAE), which is, according to Shani and Gunawardana [80], one of the metrics most often applied for recommender systems.

MAE describes the average deviation of the predicted rating of a recommendation from its actual value. Let v and v' be the rating of a recommendation q predicted by a system and given in a reference set, respectively. The MAE is specified as

$$MAE = \sqrt{\frac{1}{|R_f|} \sum_{(q,v) \in R_f} |v - v'|}$$

Since the MAE captures the error of the system-rating, the accuracy of the rating predictions of the system is higher with a lower MAE. Its domain naturally coincides with that of the corresponding rating (e.g., if the ratings range in the interval [0..1], the MAE does so, too).

2.4.3. Ranking Accuracy

Shani and Gunawardana [80] also show that there are several approaches for the evaluation of the ranking accuracy (i.e., for comparing a given ranking for a set of recommendations to a reference ranking). The so-called Normalized Distance-based Performance Measure (NDPM) proposed by Yao [88] considers the ranking as a relation of preference values. Only differences in the ranking of recommendations proposed by the system evaluated influence its value. Hence, the NDPM is also applicable for cases in which the reference ranking is only specified w.r.t. groups of recommendations (i.e., there may be recommendations that are rated equally and, consequently, also have the same ranking value).

NDPM is based on the Distance-based Performance Measure (DPM), which describes the distance between the ranking determined by a system and the one given by a reference set. The NDPM normalizes the DPM w.r.t. the worst-case (i.e., the case when all recommendations given in the system-ranking are conversely ranked in the reference ranking). Let C be the set of the pairs of recommendations that are ranked by the system, and let $C^- \subseteq C$ be the pairs of recommendations that are conversely ranked in the reference set. Further, let $C^u \subseteq C$ denote the pairs of recommendations that are ranked by the system but not in the reference set (i.e., they have the same rank in the reference set). The NDPM is then defined as

$$NDPM = \frac{2C^- + C^u}{2C}$$

Note that the NDPM penalizes twice as much a pair of recommendations conversely ranked by the system compared to one that is not ranked in the reference set. A NDPM value of 0 states that the system evaluated correctly determines every preference relation of the reference

ranking. The system is rated worse if it contains preference relations (i.e., differences in similarity values) contradicting the reference ranking. The NDPM of the worst-case is 1.

Since the calculation of the NDPM is performed regarding the topmost recommendations, it may be the case that recommendations in the system ranking are not part of the reference ranking. Similar to Ohsugi et al. [66], who add an additional rank for all those recommendations, we consider the ratings of recommendations that are not part of the reference ranking to be 0 for the computation of the NDPM.

Moreover, note that the rating values are similarity values in this work. This is because the recommendation is based solely on matching algorithms. The ranking, correspondingly, is the order of recommendations determined by the similarity values.

2.5. Conclusion

This chapter covered the theoretical background of this work. First, relational algebra was presented as a formalism to describe the semantics of queries independently of a specific implementation language. Similarity search as a methodology for comparing objects was described thereafter. In that context, index search represents an efficient means for a textual comparison based on equality of terms. For the calculation of a corresponding similarity value, a formula based on TF-IDF (term frequency-inverse document frequency), used in the Apache Lucene Project [1], can be applied. A second matching technique was introduced with schema matching – which compares tree structures. There are several individual schema matching approaches focusing on the comparison of different sorts of information. Also, methods for combining them are proposed by Do and Rahm [29], which lead to a multitude of possibilities for creating new, combined matchers. In the last section, measures for evaluating query recommendation and matching were described. The metrics presented cover the three properties usually evaluated with recommender systems: overall usage, rating, and ranking accuracy; thus, they enable a comprehensive evaluation of the quality of the query recommendation system conceived in this thesis.

After having established the necessary preliminaries in this chapter, the query recommendation problem is introduced both from an application perspective and formally, next.

3. Problem Analysis and Definition

In this chapter, the task to be solved is concretized. The first part, a presentation of the environment of the matcher to be developed, the remix BI tool [58], is followed by a detailed analysis of query matching. Requirements for the remix report matcher and formal definitions of the important concepts are given in the remainder of this chapter.

3.1. Query Recommendation in remix

The environment for the query recommendation system is remix [58], a self-service BI tool currently developed by SAP Research. remix is targeted at business users and supports them in creating and sharing reports. The report creation is substantially facilitated by automatic recommendations regarding useful data sources, adequate queries on the data, and reasonable visualizations. Figure 3.1 shows a screenshot of remix with the elements necessary to build reports (i.e., data sources and visualizations with queries) available in a parts bin on the bottom. The available elements are mined from the remix report repository and can be easily dragged into the workspace on the right. In this way, remix encourages collaboration and sharing of reports, and motivates reuse.

Figure 3.1.: View of the remix workspace

According to the elements of a report distinguished in the user interface, the internal report model of remix, shown in Figure 3.2, the so-called *reportlet*, discerns three aspects of reports: the data, the query, and the visualization. The data aspect captures the details of the data source. The query model describes the query on the data as connection between data source and visualization (i.e., it specifies the data to be used, a computation on this data, and a

projection of the result of the computation). The visual aspect represents a table or chart including various graphical features that visualizes the result of the query.

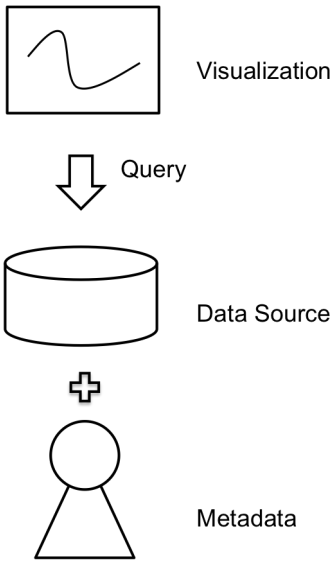

Figure 3.2.: The remix report model

In the context of report recommendation, this work will conceive a recommendation module for queries, which calculates the similarity between arbitrary queries to provide the topmost queries as recommendations. As part of this, the query aspect of the report model, which has not been specified yet, has to be developed. The latter has to capture the important elements of a query such that different matching techniques (e.g., machine learning as well as a semantic comparison) can be applied. For the similarity calculation between reports, in addition, distinctive features need to be extracted of the data and visualization model. Nevertheless, the focus of this thesis is on query recommendation by means of query matching, the main challenge to be overcome for calculating similarity of reports.

3.2. The Query Matching Problem

The main issues and challenges connected with query matching are introduced in this section. To outline them, a query matching problem is given as example together with a rather straightforward solution using common Information Retrieval techniques for text search. The challenges of query matching are described, subsequently, in more detail.

3.2.1. An Example

This section gives an introductory example, Example 3.1, for the problem of query matching together with a solution based on text search. Since state-of-the-art reporting tools apply substring matching and keyword search to match queries [50], the outcome gives an idea of their performance. Moreover, text search is the baseline approach used in the evaluation in Chapter 6.

Example 3.1. Consider the SQL queries given below as description of an exemplary problem. A query matching system then has to compare the query issued by a user, q_{orig}, to the queries stored in the system repository. As a solution, it has to deliver similarity values for the pairs

(q_{orig}, q_0),(q_{orig}, q_1), (q_{orig}, q_2), and (q_{orig}, q_3), respectively.

---

q_{orig}:
```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

q_0:
```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

q_1:
```
SELECT name
FROM dept d JOIN manager m ON d.id=m.deptid
WHERE totalsales >50000
```

q_2:
```
SELECT *
FROM department
```

q_3:
```
SELECT id
FROM manager
WHERE name LIKE 'Chong'
```

In the following, the straightforward approach text search is used to solve the problem. For that, the repository queries are indexed in complete, as text. After transforming the query of the user correspondingly, it can be used as *meta-query* to query the index (i.e., it looks for indexed queries that contain many of the tokens given in the meta-query). The meta-query for q_{orig} and the indexed query q_1 are pictured below.

---

q_{orig}:
```
TEXT:SELECT OR TEXT:name OR TEXT:sales OR TEXT:FROM OR ...
```

q_1:
```
TEXT:
SELECT name
FROM dept d JOIN manager m ON d.id=m.deptid
WHERE totalsales >50000
```

In the search process, the indexed queries are ranked according to their relevance, which can be seen as similarity measure. The result of the text search, calculated using the common TF-IDF measure, are the similarity values 1.0, 0.03, 0.14, and 0.05 for the pairs (q_{orig}, q_0), (q_{orig}, q_1), (q_{orig}, q_2), and (q_{orig}, q_3), respectively. Since the query q_0 in the repository exactly matches q_{orig}, the corresponding similarity value must be the greatest. The coarse-grain comparison based on term equality is reflected by the values for the three queries q_1, q_2, and q_3, which only slightly differ. Note that the pair (q_{orig}, q_1) is rated the least similar, although the queries have a very similar intend. On the other hand, the result of q_{orig} is contained in that of q_2. Further, the similarity calculation based on TF-IDF might be inadequate for queries in different query languages.

As a result of the textual comparison, the system could return the top three queries, q_0, q_2, and q_3, as recommendations.

The problem in Example 3.1 shows several of the difficulties of query matching introduced by the complexity of the query syntax, its semantics, and the required computation of a similarity value. Nevertheless, further challenges are introduced by system construction and integration, which are described in the following.

3.2.2. Challenges in Query Matching

The above example shows that the task of matching two queries is non-trivial. Due to the integration in remix and the peculiarities of query formulation, the matcher has to overcome several challenges:

- The body of queries and auxiliary information can become very large
- The queries to be matched and the information given are often very heterogeneous
- The query structure is complex

Since remix is intended for a long-term application, the queries in the system and the available auxiliary information considerably *grow over time*. Nevertheless, the processing must not exceed the limits of what is acceptable regarding time and memory requirements. Therefore, the complexity of the approach has to be kept within reasonable limits (e.g., schema matching usually is of quadratic or even cubic complexity, which is not appropriate for the large-scale, interactive setting of remix).

In remix, the query issued by the user is matched against the set of queries in the query repository. At the same time the queries are considered as auxiliary information to support the matching process. The *varying quantity and quality* of these queries has to be considered as well as the fact that they often come from *various tools* (i.e., they are written in a specific query language and format) and *different domains* (e.g., considering reports of different departments of one company). For these reasons, the query matcher of remix has to use the available queries efficiently and should cope with queries in different query languages and domains.

Nevertheless, the *complexity* introduced by the possibilities of query formulation presents a major challenge to be overcome in query matching. A query has to adhere to a specific query language. Additionally, a certain set of result tuples can be retrieved with different queries (e.g., with queries that overlap). Furthermore, there are different design possibilities to describe semantically equivalent expressions. A matching algorithm, consequently, has to apply techniques that resolve such syntactic and semantic inconsistencies.

The above challenges hint at the tasks to be solved by a successful query matching approach. Thus, they lead to the requirements for the query recommendation system, which are given in the next section.

3.3. Requirement Definition

The integration into remix requires the query recommendation system to work independently of a specific query language and domain. Additionally, it defines the acceptable runtime and the available memory capacity. Next to these practical issues to be considered, the challenges of query matching described in the previous section show further requirements

– quality, robustness, and efficiency – to be fulfilled by a successful query recommendation system. Hence, a collection of criteria is given, in the following, which defines the requirements for an adequate solution to the query recommendation problem in the context of the remix architecture.

C1 Integration: The recommendation system should be integrated in the remix architecture and make use of the Auto-Mapping Core (AMC) [68], a schema matching framework developed by SAP Research.

C2 Interactivity: The interactive approach of remix requires the recommendation system to proceed in an acceptable time. In concrete terms, this means that matching a query against a repository should not exceed a logarithmic complexity in dependence of the repository size.

C3 Interoperability: Due to the tool independence of remix, the developed system has to be applicable to queries in arbitrary query languages.

C4 Quality: The recommendations provided need to be correct and complete w.r.t. the queries given in the repository. The correctness requires a fine-grained comparison (i.e., including the syntactic, structural and semantic aspects of queries) and the completeness a maximal reuse of queries.

C5 Independence: The query recommendation system has to be applicable domain-independent and should produce results of similar quality in comparable use cases.

C6 Robustness: Due to the uncertainty, specificity, and scarcity of information given with the stored queries, the system has to cope with false and ambiguous information.

C7 Efficiency: The result of the recommendation system should depend on both the amount and quality of the input information. Thus, the quality of the result should reflect this of the input and enhance with augmented input.

The above criteria can be assigned to the two groups of functional requirements (C1-C5) and non-functional (C6, C7) requirements. Since the goal of this work is to develop a report matcher for remix, the integration in the remix architecture (C1) is a major requirement. It is only fulfilled satisfactorily, if the equally required interactivity (C2) and interoperability (C3) criteria are met. Nevertheless, for the system to be useful, it has to produce results of good quality (C4) in arbitrary environments (C5). Desirable qualities of the new recommendation system are robustness (C6) and efficiency (C7), which are not covered by the evaluation.

3.4. Problem Definition

After having analyzed the query recommendation and matching problem, this section provides formal descriptions for the main concepts relevant in this context – it defines the query matching and the query recommendation problem formally.

3.4.1. Query Matching

Definition 3.1 (Query Matching Problem)**.** *Given two queries q_0 and q_1, find a plausible value between 0 and 1 for the similarity between the queries exploiting all the information given by the queries themselves and auxiliary sources.*

Algorithms which solve the query matching problem by computing similarity values for queries are called *(query) matchers*. Moreover, a similarity value greater than 0 classifies two queries to be *similar* to some degree.

The major challenge of query matching is to correctly reflect the similarity of queries in the similarity value. The quality of a query matcher is determined by the correctness and completeness of its results. The former requires pairs of queries with a similarity value greater than 0 to be similar, the latter similar pairs of queries to have a similarity value greater than 0.

In this thesis, the similarity values of queries are calculated and used to rate them during query recommendation, which is specified next.

3.4.2. Query Recommendation

Definition 3.2 (Recommendation)**.** *Queries that are similar are recommendations for each other.*

Definition 3.3 (Query Recommendation Problem)**.** *Given a query called original query, q_{orig}, and a set of queries Q as possible recommendations, return a sorted set $R_f \subseteq Q \times [0..1]$ of the queries most similar to q_{orig} together with the corresponding similarity values, which determine the order. The similarity thereby can be based on the queries themselves as well as on external sources.*

The queries in R_f are the recommendations finally presented to the user. In the following, often, the set $R \subseteq Q \times [0..1]$, $|R| = |Q|$ is used to describe the processing. It contains all possible recommendations and the corresponding similarity values. The number of final recommendations to be returned (i.e., $|R_f|$) is set through the size-parameter s. R_f is obtained from R by selecting the first s pairs in R.

Note that both R and R_f are assumed to be in descending order. Further, they are indexed (i.e., $R = \{(q_0, v_0), (q_1, v_1), .., (q_{|R|-1}, v_{|R|-1})\}$), and $R(i)$, $i \in [0..|R|-1]$, denotes the pair $(q_i, v_i) \in R$ at index i.

In contrast to the large body of works on recommender systems, which concentrate on *collaborative* approaches, this work follows a *content-based* approach of query recommendation (i.e., the recommendation is based primarily on the information given with the queries themselves). This is also emphasized by the fact that the recommendations are augmented with similarity values – recommendations in recommender systems are usually annotated with so-called *predictions* or *ratings*.

The challenges of query recommendation are three-fold: obviously, the recommendations finally presented to the user have to be recommendations for the query of the user. In addition, the corresponding similarity values have to be correct, and the ranking of the recommendations has to make sense. Since the definition above requires the similarity values to be of a specific interval, they serve to determine the ranking, in the following.

This work solves the query recommendation problem in the way induced by the above definition, through calculating similarity values for the queries and proposing the most similar queries as recommendations.

3.5. Conclusion

This chapter described the query recommendation and matching problem in detail. It, first, presented the reporting tool remix as the environment of the query recommendation system to be developed. Thereafter, an example demonstrated the task to be solved and outlined the challenges connected with query matching. In particular, the amount of information, its heterogeneity, and the complexity of the queries have to be overcome. Based on these considerations, the requirements for the query recommendation system were specified. Altogether, the query recommendation system has to be integrated in remix and provide recommendations of good quality. In the end of the chapter, the problems of query recommendation and matching were defined formally.

After the detailed analysis of the problem to be solved and its specification, related work in query recommendation and matching is reviewed in the next chapter. Especially, the requirement definition presented above serves to demonstrate the features still missing with existing query recommendation approaches.

4. Related Work

This chapter presents related work in the fields of query recommendation and matching. The first part introduces existing approaches of query recommendation, gives an overview of query matching in the area of query optimization and hints at further applications of query matching. The second part of this chapter highlights techniques in two other domains of matching, similarity search and schema matching, which can also be applied for matching queries. Finally, a classification of query matching approaches integrating the existing works and providing the basis for a detailed comparison is derived.

4.1. Query Recommendation and Matching

The central topic of this thesis is query matching for query recommendation. Nevertheless, query matching is performed with very different goals, too. This section first presents the works closest to this study, which target query recommendation as add-ons for Database Management Systems (DBMS). Then, the origins of query matching, which lie in the fields of data integration and optimization (i.e., in the internals of DBMS) are described together with current works in these areas. Lastly, other applications of query matching are outlined.

4.1.1. Query Recommendation

Providing automatic recommendations of similar queries to a user who has issued a query is a very recent area of research. DBMS, generally, do not offer further functionality in this direction; however, they often maintain query repositories [8, 14] or query logs [4, 5, 52] where the user can manually search for relevant queries. Such repositories are shortly covered in the beginning of this section. Recommender systems [16, 22, 37, 38, 49, 77, 78], which provide query recommendations after a query has been issued by a user, are studied thereafter. Similarly, query completion mechanisms [47, 51, 87] are described, which propose completions for a partial query formulated by a user.

Query Repository

A stored collection of queries that is accessible to the user presents a very simple query repository (i.e., without further infrastructure functionality) and can be seen as a very basic form of query recommendation. Current DBMS usually maintain such a repository for one of two reasons: either to provide a selection of well-defined query examples [8, 14] often annotated with natural language descriptions or to store the querying history in a so-called *query log* [4, 5]. However, both approaches do not represent very elaborate query recommendation techniques. Although providing a selection of high-quality queries, the whole of example queries can never fit to the query a user is looking for. Query logs, similarly, do not allow for pre-selections w.r.t. a query a user is looking for; they usually target physical tuning. Although the advantages of an enhanced infrastructure for query repositories were already described by Khoussainova et al. [50], they have not been adopted, yet.

In [52], Khoussainova et al. present a query log with enhanced browsing functionality:

The user has access to all queries issued in the past, which are organized session-based[1], and can search for relevant sessions with keywords identifying query parts (e.g., employee AND departments). But the focus of Khoussainova et al. is on the positive impact of the session-based browsing functionality itself, the identification of relevant queries in the sessions is based on rather low-level matching techniques, the text search and ranking capabilities provided by the underlying DBMS.

Recommender Systems

In their goal of providing recommendations to the user after he has issued a query, recommender systems are very close to this work. However, these systems are, per definition, based on a collaborative approach[2]. The information contained in the queries themselves is only used marginally to match them and only by a subset, the so-called content-based recommender systems. Next to relational systems [16, 22, 49], there are several multidimensional recommender systems [37, 38, 48, 77, 78]. A good overview of the state of the art is provided by Marcel et al. [60, 61].

With YMAL, Stefanidis et al. [49] propose three techniques for the recommendation of query results called *current state*, *history*, and *external sources*. The three approaches concentrate on different kinds of information available in a DBMS – the query issued by the user and the database, the query log, and arbitrary data sources, respectively – but only with regard to query results. Next to a result-based approach, the QueRIE system [16, 22] integrates a fragment-based computation scheme for query similarity. Its goal is to determine queries in the query log that fit into the session of the current user. After applying relaxation methods following Koudas et al. [53], the log queries are split into fragments as illustrated in Table 4.1. Then, fragments that often occur together in a session are defined as being similar. In essence, the final recommendation includes those queries from the query log that contain many fragments from the current user session.

Fragment name	**Start keyword**	**End keyword**
Attribute string	SELECT	FROM
Relation string	FROM	WHERE, GROUP BY, ORDER BY, end of query
Where string	WHERE	GROUP BY, ORDER BY, end of query
Group By string	GROUP BY	ORDER BY, HAVING, end of query
Having string	HAVING	ORDER BY, end of query

Table 4.1.: Fragment definition in the QueRIE system, adapted according to [17]

Among the recommender systems for multidimensional databases [37, 38, 48, 77, 78], only few [37, 38, 77, 78] compare the current query to previous queries. Furthermore, this comparison is only syntactical. Sapia [77, 78] considers properties of the query as a whole (i.e., result measures, selection and result levels) and groups queries with common fragments. Likewise,

[1] In this context, a session is usually represented by a set of queries in a DBMS issued in a specific interval of time by one user.

[2] In the collaborative recommender systems, the recommendation depends on information about the behavior of other users and on preferences of the current user [76].

Marcel et al. [37, 38] resolve similar queries by comparing the references to the OLAP cube[3], which are defined through the selection and projection of the queries. W.r.t. these sets of references, the similarity value for two queries is computed using the Haussdorff distance[4]. In [36], Marcel et al. propose to use the Hamming distance[5] to compare two single references.

Although the recommender systems presented match queries, the techniques suggested are basic methods of text comparison. Since their focus is on comparing queries with a session in a DBMS (i.e., the over-all user behavior), the query comparison itself only constitutes a rather small portion of the computation and is only based on the information available in a DBMS. Another important issue is raised by Marcel et al. [61], who state that the quality of the recommendations needs to be assessed based on real users and cases, which has not been done, to date.

Query Completion

Another service on top of DBMS is provided by query completion approaches that aim at completing partial queries. Although several commercial systems provide auto completion regarding tables, columns, and functions [3, 6], only few and rather recent works [47, 51, 87] cover a more elaborate completion.

Already Ioannidis and Viglas [47] emphasize the need for an internal representation of queries and describe a special parse tree as a normalized query representation. They concentrate on the use case that a complete first query is incrementally refined by a user. The refinements may be simple phrases, which are automatically adapted to the context of the original query. The so-called *conversational querying* happens interactively and also previous, stored queries can be included in the context search performed by the system. The context search is one major contribution of the work. However, the proposed algorithms consider only select-project-join (SPJ) queries (i.e., no queries with aggregations and no nested queries) and for the final completion only one query is used – the information contained in the remaining queries is neglected. Yang et al. [87] automatically complete a query with join operations (i.e., join tables and join conditions), given start and end attributes. The paths of joins are found by mining possibilities from the query log; for that, a greedy algorithm is applied, which maximizes a predefined quality measure (e.g., the share of queries in the log that contain the selected path). In contrast to Yang et al., who only resolve join relations, Khoussainova et al. [51] recently presented an interactive query completion method, which takes arbitrary parts of the query (e.g., tables, selection conditions, and aggregates), so-called *query snippets*, into consideration. By keeping the queries from the query log in a directed acyclic graph (DAG), similar to the example shown in Fig 4.1, which merges common query parts (i.e., the nodes contain sets of snippets and a query is represented by exactly one node), valid and probable completions can be recommended to the user.

[3] In Online Analytical Processing (OLAP), the data model considered is often multidimensional, and can, then, be represented as a cube. This cube is precomputed as view of the database, and queries usually reference that cube instead of directly addressing the database tables [9].

[4] The Haussdorff distance is a measure from set-theory. It calculates the distance between two sets based on the distance of the corresponding elements. In essence, two such sets are close, if the pair-wise distances between the elements of the two sets are small [46].

[5] The Hamming distance is a string metric. It compares two strings by adding up the differences between symbols at equal positions in the strings. It is defined as $d_{hamming}(\langle a_1..a_n\rangle, \langle b_1..b_n\rangle) = \sum_{i=1}^{n} equals(a_i, b_i)$, with function $equals$ being 1 if a_i and b_i are the same and 0 otherwise [44].

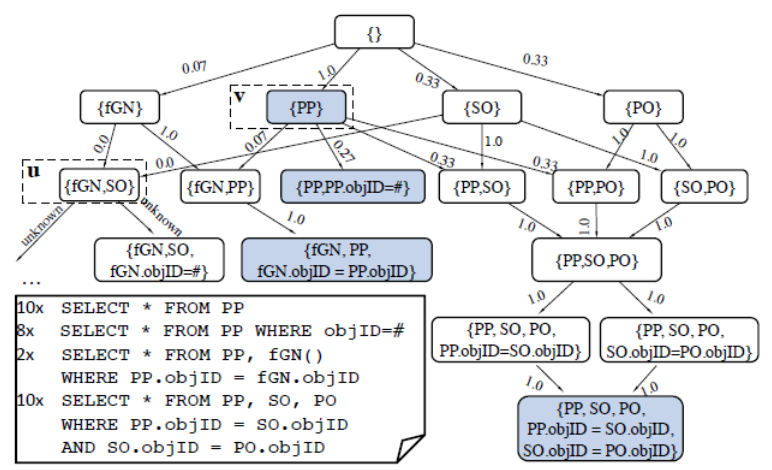

Figure 4.1.: The queries of a query log, shown below left, coalesced in a DAG by merging common query parts; taken from [51]

With the suggestion of graphs for query representation, the approaches for query completion explicitly take into account structural inconsistencies between queries. However, the graphs will be considerably greater in size and more complex, if queries of arbitrary complexity and data sources are considered (e.g., in a context more general than one DBMS). Therefore, the matching methods proposed by [47, 51, 87] need to be adapted in a more complex setting.

4.1.2. Query Optimization

Computing the similarity between queries is a major challenge in query optimization; primarily, to make use of materialized views [43, 89]. The field of multi query optimization (MQO) [33, 74, 79, 86, 91] coalesces multiple similar queries to minimize the database request. Furthermore, similarities between queries are determined to reuse existing query execution plans. This section gives an overview of query matching in these three areas.

In line with the practical relevance of the query matching problem for query optimization, several works have been proposed to solve it on the theoretical level. Early research [42, 72] as well as recent work [24, 63] focuses on the algorithmic aspect of the problem, where the goal is to decide about the relationship between queries. Since the algorithms published in these papers are primarily exhaustive (i.e., consequently, they are of exponential complexity) and usually only consider special cases of the problem, they are of minor relevance for matching complex queries.

Materialized Views

Materialized views (i.e., precomputed queries) are commonly used to optimize query processing. For that, incoming queries, targeting the database, have to be rewritten to include the views. Although many works consider this rewriting – an exhaustive survey is provided by Halevy [43] –, their impact on query matching is only minor. This is because most approaches are cost-based or cover only a subset of queries; as a consequence, the algorithms proposed are too simple and not suited for a more complex setting. Only Zaharioudakis et al. [89] describe an approach that works for nested queries and queries with aggregations, too. They represent the queries as graphs, which are matched by a custom algorithm supported by a list of patterns. With the need of a fine-granular query representation together with matching algorithms that

work in a piece-by-piece fashion on the one hand (i.e., they do not match the queries as a whole, but perform a fine-grained comparison based on the query in parts, the individual nodes in the graph), and translation (i.e., normalization) algorithms on the other hand, Zaharioudakis et al. [89] explicitly state the extensions that are necessary to match complex queries.

A special kind of materialized views is created by the so-called *semantic caching*. It exploits the semantic information (e.g., the selection and projection) present in query specifications to organize and manage the cache – usually a client cache in distributed applications – dynamically. Thus, the focus of existing work is more on efficient query processing and cache management (e.g., cache replacement policies) than on representation issues. Specifically, the approaches usually treat only a subset of queries [26, 62, 71, 73, 83] and are often based on Deshpande et al.[27], who describe a fix cache structure according to which the cached data is organized.

Multi Query Optimization

Another approach of query optimization is multi query optimization (MQO), which aims at identifying common subexpressions in workloads of related queries to execute common operations only once. Early works in MQO [33, 79] only regard a subset of queries and propose exhaustive algorithms that do not satisfy the efficiency requirements of the practice. For that reason, more recent studies propose to use greedy algorithms based on heuristics to optimize the search [74, 86, 91].

Roy et al. [74] extend an optimizer to detect common subexpressions and cases of subsumption, but concentrate on the cost-based selection of an execution plan. Zhou et al. [86, 91] abstract the original queries to certain properties of the queries (e.g., a boolean value identifying a grouping expression together with all source table names as in [91]). The abstraction enables a fast filtering of queries that have no common subexpressions. The common expressions are then merged and an execution plan is created. With the reduction of queries on specific properties, Zhou et al. propose an efficient means to filter out queries that do not have common subexpressions. For query optimization, which requires equality regarding subexpressions, this filter is appropriate. For query recommendation, however, total equality is not necessary. Hence, the filter would have to include further properties to enable a more fine-grained decision.

Query Execution Plan Selection

Also the selection of query execution plans can be optimized by matching queries – either to reuse existing execution plans (i.e., plans that were created for similar queries) or to execute similar queries together, which is related to MQO. Research in this field goes in the direction of query clustering [35, 40].

Gopal and Ramesh [40], however, describe an exhaustive clustering algorithm for SPJ queries, which is cost-driven. Also Ghosh et al. [35] target the clustering of SPJ queries. For the cluster assignment, they use a classifier based on decision trees[6]. The classification depends on several properties of the query (e.g., the number of join predicates in which a particular table is involved), which are more specific than those mentioned before. However, most of

[6] Decision tree learning is a machine learning technique that classifies input based on a number of characteristics. Thereby, the classification depends on a tree-model learnt in advance [55].

the properties are based on physical aspects of the data source, information that must be provided in addition to the queries for the matching to work.

4.1.3. Others

Apart from query recommendation and optimization, different kinds of queries (e.g., simple phrases, natural language sentences, or database queries) are completed, compared, and translated in several areas. Examples are fields such as phrase completion in keyword search [56], data integration [43], and natural language interfaces to databases [39, 54, 65]. Since the focus of those areas is not on determining similarity of arbitrary, complex database queries, they are not considered further in this work.

4.2. Matching Techniques

The query matching approaches described above all focus on certain aspects of the queries by abstracting from others. A more general consideration of queries also makes it possible to apply methods that have been elaborated and proven over time. This section presents two rather general matching techniques, similarity search and schema matching, that can be instantiated to query matching.

4.2.1. Similarity Search

As mentioned above, query recommendation can be seen as search problem where keyword search is not appropriate. However, a different kind of search, similarity search can be applied instead. In the context of query matching, however, only few works consider similarity search [23, 35, 86]; in addition, they apply machine learning [23, 35] or a very coarse-grained comparison [86]. But the application of machine learning requires an adequate set of samples to learn from. Furthermore, the dimensions of comparison considered in those works are not apt for a general comparison of queries since they reduce queries to very few properties. The finding of appropriate dimensions of comparison, which are able to capture the syntactic and semantic information of a query, as well as the development of a suitable similarity measure are important issues to be addressed. To the best of our knowledge, these questions have not been studied in more detail, to date.

4.2.2. Schema Matching

Since a query can be represented by its parse tree, query matching can be seen as an instance of the more general class of schema matching. Schema matching is an important research area in the database community, which contains a large body of work on various matching approaches. Schema matching systems apply linguistic methods, structural analysis, domain knowledge and past mappings in order to create a mapping between to schemas. There are several matching systems [19], today, that could be applied for matching queries, too. However, for query matching the existing systems would have to be adapted: the missing full automation of the matchers, which is caused by a lack in preciseness and completeness, and their complete ignorance of the semantics of the query language would otherwise lead to unsatisfactory results.

Differing, amongst others, in their goals, application domains, and methodologies, the query matching approaches presented in this section are largely heterogeneous. Therefore, their most important commonalities and differences should be pointed out in a comparative discussion.

4.3. Discussion

A discussion of related work coming from different domains benefits from a common vocabulary. This section, first, develops such a vocabulary as a classification for query matching approaches, which does not exist, to date. Based on this classification, the query matching approaches presented in the previous section are compared, afterwards.

4.3.1. Classification

Generally accepted classifications for schema and ontology matching approaches are given in [32, 70]. Such a uniform vocabulary and organization represent the basis for a comprehensive discussion of different matching techniques. For query matching, however, such a common foundation does not exist, yet. Based on existing work in matching [32, 70], the granularity of the approach and the information included for matching present the two central dimensions to classify query matching approaches in this study. W.r.t. the subjects to be matched in this thesis, queries, however, the classification should be adapted on the lower-levels. This section proposes a classification for query matching approaches, which is illustrated in Figure 4.2. Further, the classification proposed is considered in the context of the closest existing classifications [32, 50, 70].

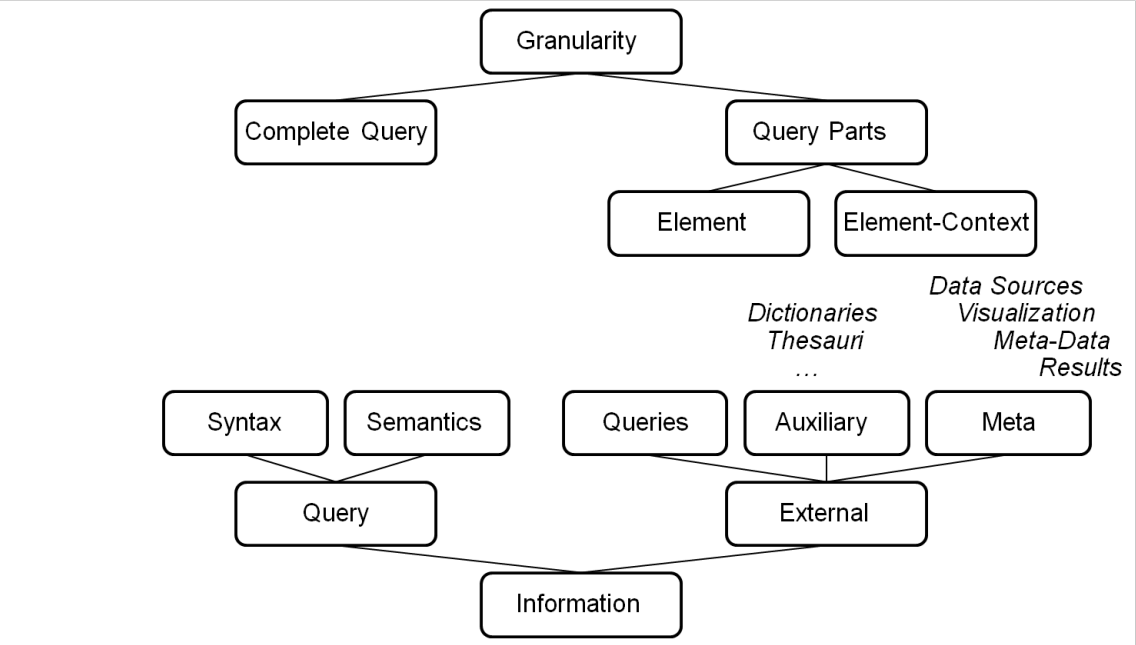

Figure 4.2.: Criteria for classifying query matching approaches

Granularity A matching algorithm processes the queries with a certain granularity considering either the *complete query* (e.g., with substring matching and keyword search) or breaking them into *query parts* (e.g., by organizing individual terms in a graph). In the second case, with *element* and *element-context*, a further distinction can be made depending on whether all elements are considered individually or in the context of the structure (e.g., elements appearing together in the structure may be matched in combinations).

Information Matching approaches can be distinguished by the information they use for matching. This information may come from the *queries* themselves or from *external*

sources. In the first case, purely *syntactic* approaches should be discerned from *semantic* techniques that include query semantics. In contrast to a simple syntactic consideration of queries, a semantic approach connects the syntactic elements of a query language with certain semantics (i.e., it interprets the queries). Matching based on external information contains various approaches, which can include meta-data, the data sources, the query results, or a visualization as *meta information* about a query. Note that this are exactly the elements that make up a report. Hence, report matching in this study will focus on the queries and may include further information contained in the reports as external information. Apart from reports, external information sources are divided into other *queries* (e.g., the use of a query log), the major information source for the so-called *reuse-based* matching [28], and other *auxiliary sources* (e.g., dictionaries or thesauri).

Rahm and Bernstein [70] introduce the basic classification for schema matching approaches. They explicitly mention possibilities to combine individual match approaches and additionally discern *instance/contents-based* techniques. W.r.t. query matching, such instances do not exist. However, query results and other kinds of external information may be considered. Because of the textual nature of queries, the above classification contains the granularity of considering queries in complete next to the element and element-context-based approaches (i.e., *element* and *structure-level* in [70]). Finally, the use of a specific query language automatically constrains query syntax and makes query semantics more concrete, which is why the *constrained-based* criterion proposed by Rahm and Bernstein is called semantic in this work. Euzenat and Shvaiko [32] extend the classification of Rahm and Bernstein by explicitly including a *semantic* dimension in their classification of ontology matching approaches, which makes sense since the nature of ontologies supports reasoning.

In a context similar to query matching, namely *meta-querying* (i.e., the search for queries), Khoussainova et al. [50] discern three different kinds of information about queries, *query-by-parse-tree*, *query-by-data*, and *query-by-feature*; however, their classification does not provide clear dimensions and is partly ambiguous. Query-by-parse-tree, targets at structural properties of the complete query (e.g., number of joins in the query). Query-by-data designates the search for queries by setting conditions on the results of the queries. Finally, the query-by-feature paradigm collects features of a query, which can be syntactic (e.g., the relation names in the query), semantic (e.g., cardinality of results or samples), or meta-data. Hence, query-by-parse-tree overlaps with the syntactic query-by-feature approach for both regard the query syntax. Although the former focuses on predefined language elements whereas the latter considers arbitrary values, the commonality of the approaches should be stated more clearly. Similarly, the semantic query-by-feature category includes query-by-data because it covers both statistics and samples. Note that Khoussainova et al. call the result-based approach semantic, which is common in related work, because the query semantics are determined by the result. In this study, the term semantic is usually used to hint at the interpretation of expressions in the context of the representation language, in contrast to a purely syntactic consideration.

In summary, the classification given in this section supports the categorization of existing query matching approaches and is, at the same time, open to integrate new techniques; especially, the granularity criterion puts no restrictions on the structure and the dimension of external information is open to include various sources of information. Moreover, the possibilities for query matching given with the above criteria do not exclude each other. As a consequence, methods can also combine different concepts of one of the two dimensions.

Approach	**Complete Query**	**Element**	**Element-Context**
Query Repository [4, 5, 8, 14, 52]	x		
Recommender Systems [16, 22, 37, 38, 48, 49, 77, 78]	x	x	x
Query Completion [47, 51, 87]			x
MVS [89]			x
MQO [33, 74, 79, 86, 91]			x
EPS [35, 40]	x		

Approach	**Syntax**	**Semantics**	**Queries**	**Auxiliary**	**Meta**
Query Repository	x		x		x
Recommender Systems	x		x	x	
Query Completion	x	x	x		x
MVS	x	x		x	
MQO	x	x			
EPS	x				x

Table 4.2.: Classification of existing query matching approaches

4.3.2. Comparison

A general comparison of existing query matching approaches w.r.t. their quality is hard. This is because they have very different goals, which often depend on specific environments (e.g., physical requirements). Nevertheless, a conceptual comparison is possible and presented in this section; it includes the query matching approaches in the areas of query recommendation and query optimization, which were described above. The classification criteria developed in the previous section thereby prove themselves by emphasizing important commonalities and differences between the approaches.

Table 4.2 classifies the approaches that target query recommendation (i.e., query repositories, recommender systems, and query completion systems) and those that target query optimization (i.e., approaches applying materialized views (MVS), MQO, or query execution plan selection (EPS)) described above w.r.t. the classification developed in Section 4.3.1.

Query repositories and query logs store the queries as they are, probably augmented with meta-information. If available, infrastructure functionality allows simple text search (i.e., it considers the query syntax) including the meta-information as external source. The various recommender systems consider either the query in complete or separate fragments (i.e., query parts). Furthermore, they include the queries from the current and earlier sessions to decide about query similarity. Often, other kinds of auxiliary information (e.g., a user profile or expectation functions) are used, in addition, as it is shown in the survey of Marcel and Negre [61]. All query completion systems considered maintain an internal graph representation (i.e., they regard the different parts of the query in their context) based on the query syntax. Sometimes, query semantics are included (e.g., in case the graph representation is normalized). To complete a query, they use stored queries as information sources and information related to specific sessions.

The only relevant approach utilizing materialized views is provided by Zaharioudakis et al. [89], who consider the parts of the query in their context by translating the query in an internal graph. The algorithm proposed for matching the graphs (i.e., the query syntax) comprises query semantics by making use of predefined patterns, an external information source. Also, the MQO methodologies model the queries as graphs. The simple syntactic comparison is often augmented with semantic considerations (e.g., by applying normalization methods) to come to a more fine-grained decision regarding the relation between the queries to merge. Lastly, the described methods for selecting query execution plans provide an example for approaches that consider several sources of information. Next to the syntactic properties of a query, they consider meta information related to the query (e.g., properties of the data sources).

The comparison shows that the consideration of queries and the information used for matching varies with the different applications. All approaches make use of the syntactic information provided by the queries themselves. Often, query semantics are considered additionally – either by using a normalized representation or by making use of predefined patterns. Next to the query information, most approaches use external information for matching. However, contrary to the combination of information sources, no approach regards the queries at different kinds of granularity, in parallel.

4.4. Conclusion

Altogether, the descriptions and the comparison give a detailed overview of the different approaches of query recommendation and matching and show that the requirements of query recommendation for remix (see Chapter 3) are only partially supported by current query matching approaches. Moreover, the evaluations of query recommendation systems have to be regarded critically. Therefore, the fulfillment of the requirements and the evaluations are considered specifically, in this section.

Requirements In general, the requirements of remix are different from those of existing systems. The complexity of most query completion approaches, a result of their detailed processing of the queries, would certainly lead to problems with interactivity in a large-scale setting. Similarly, schema matchers have not been designed to match such amounts of queries. The query matchers of recommender systems, which usually regard the queries as text, will struggle to meet the interoperability criterion – at least without a drop in quality. This is because different query languages are not matched comprehensively. The lack of quality as a result of using text comparison is shown in the evaluation in Chapter 6, which includes text search as baseline approach. Also, most of the query completion systems and query optimization approaches were developed as add-ons for one database system. Obviously, adaptations would be necessary to apply them in a more heterogeneous environment. The required quality of the approaches and also the domain independence needs to be asserted regarding their evaluations.

Evaluation Evaluations in the field of query optimization are usually executions of public benchmarks and target goals rather different from the quality of the internal query matching (e.g., DBMS related issues, especially, the cost and time of query execution). Even fine-grained matching approaches as [89] are not evaluated regarding the precision and completeness of the matching. Marcel and Negre [61] explicitly mention the shortcomings of evaluations in the area of recommender systems; especially, they point out the lack of real-world data (i.e., queries issued by general database users and

user ratings stating the usefulness of recommendations). The latter indicates that common data sets based on real-world data are needed to conduct comparable and empirical evaluations. Although Ioannidis and Viglas [47] conduct a study with real users, the experimental results have to be regarded critical since the queries used in the study are more similar to learning examples than to more complex real-world instances. Khoussainova et al. [51] conduct an evaluation of their query completion approach and make use of real-world data sets. However, they do not evaluate the recommendation of complete queries or query matching. Thus, they do not include reference data explicitly stating similarity or usefulness values for the queries, which were determined by real users.

Based on several of the approaches presented above, a query recommendation system is conceived in the next chapter. Thereby, the requirements not fulfilled by the existing systems are taken into consideration, in particular. The issues concerning the evaluation are subsequently addressed in Chapter 6, which presents an empirical evaluation of the query recommendation system based on real-world data.

5. Query Matching – A Combined Approach

The requirement definition in Chapter 3 shows that a query recommendation system for remix must provide several of the features existing systems support. For that reason, it has to follow a broader approach and integrate several of the traditional techniques.

This chapter presents a new approach for query recommendation: content-based query recommendation using a combination of two different matching techniques – similarity search and schema matching. The combination of these two, rather different approaches is chosen for two reasons. On the one hand, the task description includes the application and evaluation of schema matching, which is, however, too complex to provide recommendations interactively. On the other hand, the classification presented in Chapter 4 hints at a variety of possibilities of matching queries content-based. For exploiting most of the given information, the consideration of several approaches in parallel should be beneficial.

On account of this, methods based on the two matching techniques similarity search and schema matching are adapted, extended, and integrated into a combined query matcher, which is presented next. In the sequel, the different matching approaches are described in more detail.

5.1. Query Recommendation Approach

This section describes a conceptual framework for query recommendation based on a combined approach for query matching. Figure 5.1 gives an overview of the procedure of query recommendation. Its input consists of the original query, q_{orig}. The system itself contains the set of possible recommendations, Q, in a query repository. The output is an ordered subset of the best recommendations, R_f. After a pre-processing step, the main processing is performed by the *Combined Matcher*, a matching framework.

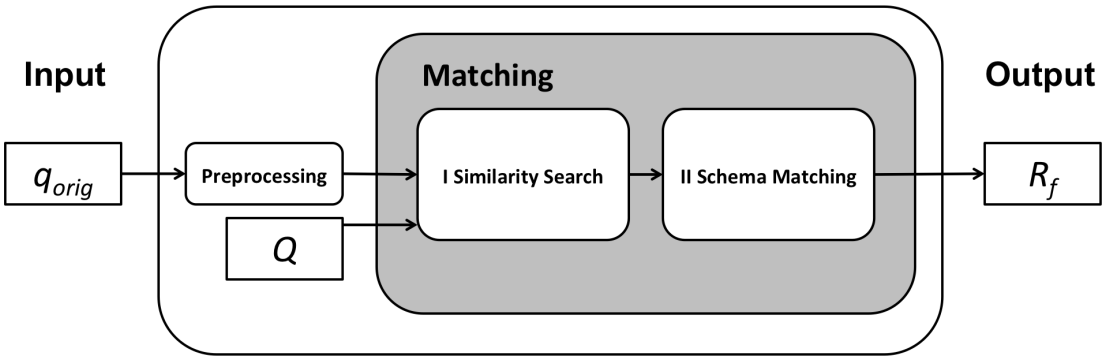

Figure 5.1.: The query recommendation procedure

The pre-processing has the goal to prepare and facilitate the matching phase. In order to enable a comparison, it parses queries in different source formats and languages and translates them into a common representation based on relational algebra. The latter is very similar to the abstract syntax tree of SQL.

The Combined Matcher, which is executed second, compares the original query to every possible recommendation and computes a similarity value based on the degree of their similarity. For that, it proceeds in two phases applying two different matching algorithms to the query pairs to be matched: the *Feature* and the *Schema Matcher*. These matchers are based on methods from similarity search and schema matching, respectively.

Each matcher recomputes the similarity value for the possible recommendations and may narrow down the set of possible recommendations by sorting out poor matches (i.e., recommendations that have a small similarity value w.r.t. the original query). After the application of the matching framework, the best matches are finally returned as recommendations.

The presentation of the global picture in this section is followed, next, by a detailed description of the Combined Matcher.

5.2. The Combined Matcher

As outlined in the recommendation procedure, the Combined Matcher executes submatchers of two kinds, similarity search and schema matching. In theory, the matchers can be applied in arbitrary order; however, in this framework every kind of matcher is applied at a particular time to fulfill a specific purpose and to overcome a particular issue of query recommendation.

The Feature Matcher based on similarity search serves as a first filtering component. This filtering is essential to guarantee efficiency if the set of possible recommendations is very large. Then, the Schema Matcher performs schema matching for a more fine-grained comparison. The schema matching helps to reflect more subtle the differences between different queries in the similarity value. Hence, the order of the final recommendations is determined.

For both matchers, concrete matching algorithms are conceived and adapted in this work. Their processing is detailed in the following. It always consists of three basic steps: the extraction of the information necessary for matching (i.e., the query content), the actual comparison, and the calculation of the similarity value. In the remainder of the section, the application of aggregation and selection strategies to enhance the result are described. To demonstrate the different concepts, the recommendation task described in Example 5.1 serves as a running example, throughout this section.

Example 5.1. As a running example, the recommendation task described below, which looks for recommendations for query q_{orig}, is to be solved by the recommendation procedure. Thereby, the SQL queries pictured below, which were already involved in the introductory Example 3.1, again, serve to demonstrate the matching algorithms and their influence on the final recommendation.

---

q_{orig}:

```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

$Q = \{$
q_0:
```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

q_1:
```
SELECT name
FROM dept d JOIN manager m ON d.id=m.deptid
WHERE totalsales >50000
```

q_2:
```
SELECT *
FROM department
```

q_3:
```
SELECT id
FROM manager
WHERE name LIKE 'Chong'
```
}

5.2.1. The Feature Matcher

Figure 5.1 shows that similarity search is applied first in the query recommendation methodology to reduce the search space of possible recommendations. At the same time, however, this filtering has to be based on a sound comparison to not filter out similar queries. This section describes how similarity values can be calculated efficiently for queries by using similarity search. Thereby, the abstract processing of the search is tailored to queries as subjects.

By considering the queries to be matched as points in a metric space, the problem of matching two queries is reduced to the comparison of arbitrary characteristics of the queries. In this case, the dimensions of the space are the kinds of characteristics, and the considered characteristics, grouped together in a vector, represent the query. The similarity of the queries then is the distance of the vectors, which can be computed using a multidimensional distance metric.

This section describes the Feature Matcher, which is based on similarity search. It introduces *features*, the characteristics of the queries that are compared. Since equality is used as basic comparison criterion (i.e., instead of a more complex comparison based on the type of the characteristics), an index can be used for the actual comparison, which is described thereafter. Finally, the similarity calculation applied on the outcome of the comparison is specified.

Definition and Extraction of Features

Globally available characteristics of queries enable a straightforward query comparison. As the query matching classification of Chapter 4 indicates, the abstraction of the textual queries onto certain characteristic properties can concentrate on different kinds of information (i.e., syntactic, semantic, or external properties). Usually, it is not appropriate and causes too much overhead to attach information in specific, external formats as the whole data source queried or the query results. The efficient use in a matching system, hence, requires the

properties to be textually recordable as so-called *features*[1]. There are various possibilities for the information recorded in features. However, w.r.t. their application for query matching based on similarity search, they differ in their usefulness.

Syntactic properties Syntactic properties (e.g., the query language, the length of the query, or the language used) are extracted from the query as it is issued in a database. Hence, they are easy to extract and naturally available for every query. But, they are not very expressive as comparison criterion and may even hint to a wrong result (e.g., although differing in their length, two queries may be similar or equal in what they deliver as result).

Semantic properties Semantic properties (e.g., the names of the data sources queried or the count of selection conditions) are based on the interpretation of the query language. A comparison based on semantic properties is closer to sound reasoning, because it reflects the actual intend of a query. Thus, it is necessary to capture the information actually contained in a query and to provide interoperability regarding different query languages.

External properties External properties (e.g., data about the user session or properties of the result set) describe the environment of the query. This kind of properties is obviously very heterogeneous and cannot be extracted with a common procedure. Furthermore, it is not available for every query and the queries differ in the external information they have attached. Thus, they do not always allow the comparison of two queries and a thorough comparison cannot be based on external properties exclusively.

For the reasons described above, this thesis concentrates on features reflecting semantic properties. Below, Table 5.1 gives an overview of all the features considered. The large number of features has been chosen in order to enable a comparison as detailed as possible.

Feature	**Description**
REL_NAMES	The names of the relations that are queried
PRO_ATTRS	The attributes in the projection
SEL_ATTRS	The attributes that occur in scope of a selection
SEL_COMP_OPS	Comparison operations that occur in scope of a selection
SEL_SQL_COND_OPS	SQL-specific condition operations (e.g., 'BETWEEN') that occur in scope of a selection
SEL_PRED_OPS	Predicates (i.e., 'and' or 'or') that occur in scope of a selection
SEL_ARITHM_OPS	Arithmetic operations that occur in scope of a selection
SEL_AGG_OPS	Aggregation operations that occur in scope of a selection (e.g., in an SQL 'HAVING' clause)
REN_RELATION_NAMES	The names of the relations that are renamed
REN_ATTRS	The attributes that are renamed
EXT_ATTRS	The attributes that are used for computing new attributes
EXT_NEW_ATTRS	The names of computed attributes
EXT_COMP_OPS	Comparison operations used for computing new attributes, the so-called *extension*
EXT_SQL_COND_OPS	SQL-specific condition operations that occur in scope of an extension

[1]Note that if a concrete query is considered, the term feature usually includes the actual value of the feature in the query.

Feature	**Description**
EXT_PRED_OPS	Predicates that occur in scope of an extension
EXT_ARITHM_OPS	Arithmetic operations that occur in scope of an extension
SORT_ATTRS	The attributes that are used for sorting the result
GRO_GRO_ATTRS	The attributes for which a grouping is performed
GRO_ATTRS	The attributes that occur in scope of a grouping
GRO_NEW_ATTRS	The names of the new attributes resulting from a grouping
GRO_ARITHM_OPS	Arithmetic operations that occur in scope of a grouping
GRO_AGG_OPS	Aggregation operations that occur in scope of a grouping
JOIN_ATTRS	The attributes that are used in a join condition
JOIN_OPS	The join operations in the query
JOIN_COMP_OPS	Comparison operations that occur in scope of a join operation
JOIN_SQL_COND_OPS	SQL-specific condition operations that occur in scope of a join operation
JOIN_PRED_OPS	Predicates that occur in scope of a join operation
JOIN_ARITHM_OPS	Arithmetic operations that occur in scope of a join operation
SET_OPS	The set operations that occur in the query
REN_COUNT	Count of rename operations
JOIN_COUNT	Count of join operations
SET_COUNT	Count of set operations
FUN	The functions that are used in the query
FUN_ATTRS	The attributes that are used as function parameters
CASE_COUNT	Count of condition statements (e.g., SQL 'CASE-WHEN' or 'IF-THEN-ELSE' statements)
SUB_COUNT	Count of subqueries
VALUES	The values that occur in the query

Table 5.1.: The query-features considered in this work

As mentioned earlier, the extraction of features based on semantic properties relies on the interpretation of the query. Hence, the queries are not considered as given (i.e., as text) but in the internal, abstract representation created during pre-processing. The features then can be extracted for all the queries to be compared. This is demonstrated in Example 5.2.

Query	*REL_NAMES*	*PRO_ATTRS*	*JOIN_OPS*
q_0	department	name,sales	
q_1	dept,manager	name	INNER
q_2	department		
q_3	manager	id	

Table 5.2.: Three features extracted for the example queries

Example 5.2. For the matching task of the running example, three kinds of features are considered: *REL_NAMES*, *PRO_ATTRS*, and *JOIN_OPS*. The outcome of the feature extraction is shown in Table 5.2. It shows the features used for comparison in the header, and the actual values extracted from the different queries in the body.

Index Search

The comparison of the extracted features can be efficiently realized with index search, which achieves logarithmic complexity. Therefore the queries to be compared are indexed using the different features occurring in them (i.e., the indexed terms do not refer to documents, but represent queries). For the comparison of two queries, the index has to be queried by transforming one of the two queries in a search query for the index. This is demonstrated in Example 5.3

Example 5.3. Below, q_{orig} has been transformed in a query for index search. The index considered uses the three features *REL_NAMES* and *PRO_ATTRS*, and *JOIN_OPS*, for indexing. This indexing of the individual queries is illustrated for q_1, which is shown how it is represented in the index.

---

q_{orig}: *REL_NAMES*: department OR *PRO_ATTRS*: name OR *PRO_ATTRS*: sales

q_1:
REL_NAMES: dept , manager
PRO_ATTRS: name
JOIN_OPS: INNER

Ranking and Similarity Computation

Since the index-based comparison relies on the text recorded in the features, the distance function has to calculate a similarity value using the corresponding information. Existing work in text search and matching often concentrates on TF-IDF [30, 31, 81]. Next to the term frequency and inverse document frequency, Apache Lucene [1], a library for index search, includes further factors in the calculation, which also provide a useful means for indexing features; therefore, an adapted version of the formula is used as a basic method for the similarity search approach. This section, first, demonstrates the similarity calculation using this formula in Example 5.4. Second, it proposes methods for refining it.

Example 5.4. Consider again the index of Example 5.3. In this example, a similarity value is computed for each query regarding q_{orig}. For the corresponding formula, refer to Chapter 2. In the following, always, $q_s = q_{orig}$. First the normalizing factor, common for all values, is calculated as

$$\begin{aligned} qNorm(q_{orig}) &= \frac{1}{\sqrt{idf(department)^2 + idf(name)^2 + idf(sales)^2}} \\ &= \frac{1}{\sqrt{(1+\ln\frac{4}{2+1})^2 + (1+\ln\frac{4}{2+1})^2 + (1+\ln\frac{4}{1+1})^2}} \\ &= 0.4022 \end{aligned}$$

Then, the individual values are calculated with function $fscore$ as for the two examples below:

$$\begin{aligned} &fscore(q_{orig}, q_0) \\ &= 1 * qNorm(q_{orig}) * (1 * idf(department)^2 + \sqrt{\frac{1}{2}} * idf(name)^2 + \sqrt{\frac{1}{2}} * idf(sales)^2) \\ &= 1.9536 \end{aligned}$$

$$fscore(q_{orig}, q_1) = \frac{1}{3} * qNorm(q_{orig}) * 1 * idf(name)^2$$
$$= 0.2223$$

After the computation of the similarity values, the filtering can be realized such that queries that are no recommendations according to the feature-based computation do not pass the filter. Further possible passing-criteria are a threshold for the similarity value or a restriction on the number of queries to pass the filter (e.g., only the best 10 recommendations are forwarded to schema matching).

Since the Lucene-based distance function allows to rank the matching queries according to their similarity, it can be applied for filtering as intended in the matching framework and described above. Nevertheless, the result values of that formula are too arbitrary for a general query matcher. First, the multitude of features considered enlarges the differences between the similarity values of the queries. Since the values, further, exceed the interval of [0..1], they are not comparable to the results of other matchers. Moreover, the values are not symmetric. To overcome these obstacles and to construct a Feature Matcher we propose three extensions, which are described in the following.

The calculation of the similarity values should focus on the common features of the queries compared. To stress these similarities, features that occur only once in the data set may be neglected in the similarity calculation (i.e., the search includes only fields for features where at least one term occurs in at least one of the queries compared that is not the search query itself). The resulting effect is that the difference between the similarity of the search query to itself and to an arbitrary query q (i.e., $fscore(q_{orig}, q_{orig}) - fscore(q_{orig}, q)$) is not enlarged by including features that occur only once in the data set. Hence, for example, the search query corresponding to q_1 would not include the *JOIN_OPS* feature, because an inner join does not occur in any other query. Thus, the similarity calculation is adapted to the current context (i.e., to the set of queries considered). This consideration is sound as long as the similarity values are considered in this context since the over-all ranking is maintained.

To enable a comparison between different index searches, the Lucene-based distance function includes a normalization factor. But to make the feature-based matchers comparable to other matchers, the distance function has to deliver a value between 0 and 1. Furthermore, the value should be the same for corresponding pairs (i.e., it must be symmetric). On the contrary, the above function computes also values exceeding the interval of [0..1] and is not symmetric. The latter is because the coordination factor and the term frequency refer to the second parameter of function $fscore$ that are not always the same for arbitrary queries. As a consequence, the value has to be normalized.

To normalize the similarity values, the similarity of the search query to itself, usually the maximum of all results, is used as a reference, and every result is divided by that value to obtain a similarity value between 0 and 1. The symmetry is obtained by taking the average of the two similarity values that should be made symmetric. This is demonstrated in Example 5.5.

Example 5.5. In Example 5.4 the similarity values were computed for $q_{orig} = q_0$. Similarly,

the values for all other queries can be computed leading to the matrix shown below.

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 1.95 & 0.22 & 0.22 & 0.00 \\ 0.16 & 1.95 & 0.00 & 0.22 \\ 1.29 & 0.00 & 1.29 & 0.00 \\ 0.00 & 0.91 & 0.00 & 1.29 \end{pmatrix} \end{array}$$

The values calculated in Example 5.4 correspond to the entries in the first row of the matrix. After adapting the matrix regarding the interval [0..1] (i.e., all values of a row are divided by the maximum of that row) and taking the average of all matrix entries that should be symmetric, the matrix looks as follows.

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 1.00 & 0.10 & 0.56 & 0.00 \\ 0.10 & 1.00 & 0.00 & 0.41 \\ 0.56 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.41 & 0.00 & 1.00 \end{pmatrix} \end{array}$$

Note that, for example, the similarity value calculated by the Feature Matcher for the pair (q_0, q_3) is 0. Since the Feature Matcher serves as a filter in the Combined Matcher, q_3 is eliminated and not considered further.

As a result, the Feature Matcher would provide the following recommendations for $q_{orig} = q_0$:

$$R_f = \{(q_0, 1.0), (q_2, 0.56), (q_1, 0.10)\}$$

Global features of a query represent a useful means for deciding about query similarity, because extracted and stored once, they are fast accessible at any time in future and they may represent arbitrary pieces of information. Nevertheless, features cannot capture the entire syntactic and semantic information contained in queries. The recommendation of the Feature Matcher shows that it detects similarities between the queries, but the calculation of the values is based only on the equality of individual terms (e.g., it does not recognize a similarity between the *sales* and *totalsales* attributes in the example). In particular, contextual information beyond the kind of the index fields is not retained (e.g., features in a subquery would not be discerned from those on the first level). For that reason, schema matching, which performs a more fine-grained comparison, is applied, as a second phase of matching – to refine the similarity values.

5.2.2. The Schema Matcher

A comparison of queries on syntax level should include a very fine-grained element-level comparison and a consideration of subexpressions of different granularity. Both cases are addressed by schema matching, which can be used for matching the parse trees of queries. Schema matching, applied second in the matching framework shown in Figure 5.1, constitutes a major component of the methodology. This is because there are many algorithms which can be adapted, extended, and used as a basis for a new Schema Matcher for queries.

By considering the queries in the form of their parse trees, the problem of matching two queries becomes a schema matching problem. Schema matching makes it possible to match individual query parts, which are groups of schema elements, of different granularity and w.r.t. several sorts of information (e.g., the subexpressions can be compared regarding their

syntax but also include semantic information by comparing data types). To determine a single, overall similarity value between two queries, the similarity values of the individual element pairs need to be aggregated after matching on element level.

In order to apply schema matching methods, the query has to be represented in a schema structure. Then, a schema matcher compares the elements of the two schemas to each other w.r.t. specific criteria (e.g., regarding the similarity of the names) and computes similarity values for the individual pairs of elements. Lastly, those similarity values are coerced into one value. This processing is described in more detail, in the sequel.

Schema Extraction

The schema structure taken in this work, in essence, is the parse tree of the queries. However, in order to abstract from different query languages, it is built from the abstract representation. The latter is augmented by making additional information (e.g., the name of the query and data types) explicit. The information added depends on what information is processed by the schema matchers (for an overview of the matchers considered refer to Chapter 2) and can be easily encoded in textual form. Figure 5.2 shows the schemas for two example queries.

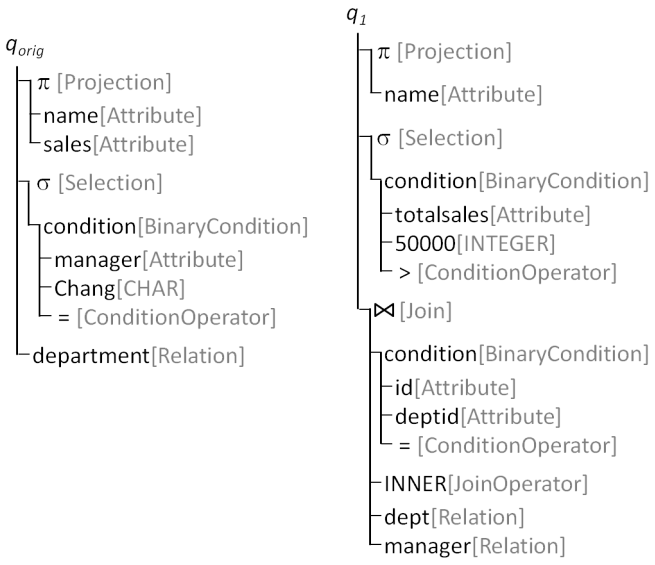

Figure 5.2.: The schemas for the two example queries q_{orig} and q_1, shown below

q_{orig}:
```
SELECT name, sales
FROM department
WHERE manager='Chang'
```

q_1:
```
SELECT name
FROM dept d JOIN manager m ON d.id=m.deptid
WHERE totalsales>50000
```

As already mentioned, the schema structure has been optimized for the processing of specific matchers. The particular points considered are described in the next section.

Schema Matchers

Unlike with matchers based on similarity search, which may compare different features but use the same algorithm (i.e., index search), schema matchers may execute very different algorithms to perform the actual comparison. An overview of several traditional schema matchers is presented in Chapter 2. Note that this are exactly the matchers available in the AMC [68], which are evaluated for query matching, in this study. This section, first, gives reasons for the application of existing matchers and details about the chosen schema structure w.r.t. the matchers evaluated. Thereafter, the processing of the matchers is demonstrated in an example.

Contrary to developing custom query matching algorithms that perform schema matching, this thesis concentrates on the application and evaluation of traditional schema matching algorithms combined in the Schema Matcher. Connected to this are three major advantages:

1. The existing schema matching algorithms have been developed over a long time and present proven solutions for the schema matching problem. They incorporate a lot of experience and cover most possibilities of matching.

2. Schema matching is a rather general classification criterion for algorithms, and there are multiple schema matching approaches, which apply different kinds of schema matchers. By concentrating on the creation of the schema structure, the query matching system is independent of a specific selection of schema matchers and can be configured according to external circumstances (i.e., considering the matchers available as well as the requirements of a particular application domain).

3. The independence of specific schema matchers makes the system lend itself to evaluations of different schema matchers regarding their applicability for query matching.

Hence, instead of adapting the existing matchers, we tailor the schema structure to their processing. The preliminaries include matchers of different granularity: element-level (i.e., the Type, Name, and Path Matcher) as well as structure-level matchers (i.e., the Parent, Children, Sibling, and Leaf Matcher). The information matched is determined by the element-level matchers applied; apart from the Type Matcher, all of them concentrate on the names of the schema elements. As a consequence, these names have to be chosen carefully during schema creation. Also, because the structure-level matchers of the AMC are based exclusively on name and type matching. Next follows a description of how the parse trees of the queries are transformed for matching, in this work.

As mentioned above, the schema structure is based on the abstract representation of the queries. A corresponding schema naturally contains a named element for every term in the algebraic expression. The term thereby determines the element-name. This schema can be enriched with further information. In particular, the type of the subexpression is added to every schema element (e.g., Figure 5.2 shows, beneath others, an element with name *Salary* and *Attribute* as type). Recall that general database schemas include type information only for attributes, the leaves of the schema.

Although the matchers considered do not regard further semantic information, this information can be used to set up the schema structure. The following points were considered for adjustments to the original schema structure, resulting from the parse tree:

- In the algebraic expression, the operations are applied one after another (e.g., in Figure 5.2, the selection nodes, named σ, would be child nodes of the projection nodes, denoted by π). Often, however, the order has no influence on the semantics of the complete query. Therefore, an extra node is added for operations referring to the same relation, which groups those operations together with the corresponding relation.
- Since the matchers do not consider query semantics, qualifiers (e.g., for tables or attributes) are neglected in the schema structure. In particular, the matchers do not recognize the relations within one query that are established by qualifiers, which relate the attributes to the relations they refer to. Next to the uselessness of the qualifiers, another reason for their removal is that they otherwise can be matched falsely (i.e., a matcher maps them to other query parts) and then degrade the result.
- Similar to the removal of qualifiers, rename operations are not considered – although they may contain expressive names and, hence, present a source of information. This is because renaming is usually applied to facilitate query writing. As a consequence, the names introduced are often very short and rather cryptic such that a degradation of the result is more likely than an improvement.

Another kind of semantic information is that introduced by the query language, or algebra, with its expressions (e.g., the name of a function could be explicitly added as a child of a *Name*-element, which, itself, is the child of a *Function*-element – instead of adding the name directly as a child of the *Function*-element). This would add considerably more information to the schema, which could be exploited by the matchers. Especially, the preciseness of the Path Matcher would increase because the paths become much more concise. In this work, however, this information is not attached. This is because several of the matchers applied are structural matchers that consider as context two consecutive levels (e.g., the Sibling Matcher regards the siblings of a schema element w.r.t. the direct ancestor of that element). Hence, these matchers would fail in establishing relations between the leaves, which are the important elements to be matched. Instead, they would consider relations between leaves and elements of the algebra (i.e., the operations), or only between the latter.

Although the different schema matchers base their similarity calculation on a custom comparison algorithm, they all return a matrix as result. This matrix contains the pair-wise similarity values for the elements of the two queries matched. Example 5.6 gives an example for such a match result.

Example 5.6. For an example of the processing of schema matchers, the Name Matcher is considered. The execution of the algorithm applied to the schemas representing q_{orig} and q_1 leads to the following matrix of similarity values between the elements in the two schemas, which are enumerated in depth-first order:

	π	*name*	σ	*condition*	*totalsales*	50000	$>$	$\bowtie$	*condition*	...
π	1.00	0.11	0.60	0.50	0.00	0.00	0.10	0.30	0.50	...
name	0.11	1.00	0.12	0.12	0.20	0.00	0.00	0.17	0.12	...
sales	0.10	0.40	0.33	0.00	0.56	0.00	0.00	0.00	0.00	...
σ	0.60	0.12	1.00	0.44	0.09	0.00	0.11	0.22	0.44	...
condition	0.50	0.12	0.44	1.00	0.10	0.00	0.11	0.33	1.00	...
manager	0.00	0.43	0.00	0.11	0.30	0.00	0.14	0.00	0.11	...
$'Chang'$	0.10	0.20	0.11	0.11	0.10	0.00	0.20	0.20	0.11	...
$=$	0.10	0.22	0.11	0.00	0.10	0.00	0.00	0.00	0.00	...
department	0.10	0.30	0.30	0.20	0.10	0.00	0.13	0.10	0.20	...

Note that the matrix contains additional matches between element-pairs, which are not displayed due to space restrictions.

The different element-level similarity values then have to be combined, which is described next.

Schema Similarity Value

Based on custom algorithms, schema matchers compute similarity values for all pairs of elements between the two schemas matched. The formula finally combining the individual values to one over-all similarity value is shown below.

$$ssim(S,T,M_{S,T}) = \frac{\sum_{i=0}^{i=|S|}\sum_{j=i}^{j=|T|} m_{i,j}}{\sum_{i=0}^{i=|S|}\sum_{j=i}^{j=|T|} sgn(m_{i,j})} * \frac{\sum_{i=0}^{i=|S|} sgn(\sum_{j=0}^{j=|T|} m_{i,j}) + \sum_{j=0}^{j=|T|} sgn(\sum_{i=0}^{i=|S|} m_{i,j})}{|S|+|T|}$$

The schema similarity function *ssim* computes the average of those values that are not zero and rates it according to the percentage of the respective elements considering all elements of the two schemas. Its input consists of two ordered sets S and T of the elements of the two schemas and a matrix $M_{S,T}$ with all the similarity values for the element pairs between the two schemas. The matrix is indexed with the positions of the elements in the sets (i.e., m_{ij} denotes the similarity value between the elements at positions i and j in S and T, respectively). Further, *sgn* stands for the Signum function, which is 0 for values that are 0 and 1 for values greater than 0. Note that *ssim*, in essence, corresponds to the function for a combined schema similarity proposed by Do [28]. This formula was adapted to capture undirected matches and instantiated with a selection function that takes the average of the similarity values of all match candidates.

Further, note that the function *ssim* assumes the similarity values returned by the matchers to express real correspondencies between schema elements. This assumption is common in the schema matching domain. In contrast, similarity values in query matching express a more subjective similarity. A query similarity value does not necessarily mean a complete correspondence of two queries if it is greater than 0.

In order to discern the real correspondencies in the result of a matcher, selection strategies as described by Do [28] are applied. A detailed description of several such strategies is given in Chapter 2. In the following, we focus on the Threshold strategy, which sets all similarity values below a specific threshold to 0, to select the real correspondencies. The threshold applied is called *combination threshold*. An exemplary application of this strategy and the subsequent calculation of the combined schema-similarity is given in Example 5.7.

Example 5.7. Consider the similarity matrix between the schemas of q_{orig} and q_1 which is computed by the Name Matcher as described in Example 5.6. The application of the Threshold strategy on this matrix using a combination threshold $t = 0.7$ results in the matrix

shown below.

	π	*name*	σ	*condition*	*totalsales*	50000	$>$	$\bowtie$	*condition*	...
π	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
name	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
sales	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
σ	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	...
condition	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00	...
manager	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
$'Chang'$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
$=$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
department	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...

The above matrix contains additional matches between the element-pairs $(department, dept)$, $(manager, manager)$, and $(=, =)$ all rated 1.0, which are not displayed due to space restrictions.

Within the above matrix only 8 entries are matches. Apart from the *condition*-element in q_{orig}, every element appears in only one match. Thus, 7 elements of the source and 8 elements of the target schema, take part in the mapping. Let $E_{q_{orig}}$ and E_{q_1} denote the sets of elements in q_{orig} and q_1, respectively. Then, the corresponding schema-similarity value for the queries is computed as:

$$\begin{aligned} ssim(E_{q_{orig}}, E_{q_1}, M_{q_{orig},q_1}) &= \frac{1.0 + 1.0 + 1.0 + (1.0 + 1.0) + 1.0 + 1.0 + 1.0}{8} * \frac{7 + 8}{9 + 15} \\ &= 0.625 \end{aligned}$$

After having computed the similarity values with the Name Matcher as above, the Combined Matcher would provide the following recommendations for $q_{orig} = q_0$:

$$R = \{(q_0, 1), (q_1, 0.63), (q_2, 0.18)\}$$

The restricted syntax of queries together with their belonging to the database domain makes schema matching a useful technique for matching queries. The problems of schema matching, which usually struggles with very large and heterogeneous real-world schemas, are narrowed down since queries are of manageable size. Moreover, the traditional schema matching algorithms offer a fine-grained processing tailored to database schemas, which resemble the query-structure. The various schema matchers also consider external information (e.g., thesauri and dictionaries). However, to benefit from the many existing schema matching algorithms, the results of different matchers have to be aggregated, which is described in the next section.

5.2.3. Aggregation and Selection

In schema matching, where a number of matchers is used to compute different similarity values for all elements in two schemas, strategies to aggregate the results of different matchers and to select specific element-to-element matches based on their similarity are commonly used [69]. Chapter 2 gives an overview of selected examples.

The aggregation enables the parallel execution of different matchers by aggregating their results. The selection is applied to the result of a matcher as a post-processing and sets similarity values that do not adhere to the corresponding selection strategy to 0. Hence, it refines the result. In this work, these strategies are applied in two ways. They are used during schema matching as originally intended [28]; the selection already has been demonstrated in Example 5.7. Additionally, the concepts are transferred to the level of queries. Both is described in the following.

In schema matching, the aggregation and selection of similarity values is on the level of elements. Example 5.8 demonstrates the aggregation. Thereby, the selection can be applied at any stage (i.e., before or after the aggregation). As mentioned above, it is important for the concept in this thesis because it strongly influences the function *ssim*, which computes the final query similarity.

Example 5.8. Consider the extracts of match results proposed by the Name and Type Matcher as shown below. The aggregation of the two match results using the Average strategy (i.e., it takes the average of the similarity values for each pair of elements) is displayed thereafter. It would then be combined to a schema similarity value as it was demonstrated in Example 5.7.

The result of the Name Matcher:

	π	*name*	σ	*condition*	*totalsales*	50000	$>$	$\bowtie$	*condition*	...
π	1.00	0.11	0.60	0.50	0.00	0.00	0.10	0.30	0.50	...
name	0.11	1.00	0.12	0.12	0.20	0.00	0.00	0.17	0.12	...
sales	0.10	0.40	0.33	0.00	0.56	0.00	0.00	0.00	0.00	...
σ	0.60	0.12	1.00	0.44	0.09	0.00	0.11	0.22	0.44	...
condition	0.50	0.12	0.44	1.00	0.10	0.00	0.11	0.33	1.00	...
manager	0.00	0.43	0.00	0.11	0.30	0.00	0.14	0.00	0.11	...
$'Chang'$	0.10	0.20	0.11	0.11	0.10	0.00	0.20	0.20	0.11	...
$=$	0.10	0.22	0.11	0.00	0.10	0.00	0.00	0.00	0.00	...
department	0.10	0.30	0.30	0.20	0.10	0.00	0.13	0.10	0.20	...

The result of the Type Matcher:

	π	*name*	σ	*condition*	*totalsales*	50000	$>$	$\bowtie$	*condition*	...
π	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
name	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	...
sales	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	...
σ	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	...
condition	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00	...
manager	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	...
$'Chang'$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...
$=$	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	...
department	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...

The aggregation of the above match results using the Average Strategy:

	π	*name*	σ	*condition*	*totalsales*	50000	$>$	$\bowtie$	*condition*	...
π	1.00	0.06	0.30	0.25	0.00	0.00	0.05	0.15	0.25	...
name	0.06	1.00	0.06	0.06	0.60	0.00	0.00	0.08	0.06	...
sales	0.05	0.70	0.17	0.00	0.78	0.00	0.00	0.00	0.00	...
σ	0.30	0.06	1.00	0.22	0.04	0.0	0.06	0.11	0.22	...
condition	0.25	0.06	0.22	1.00	0.05	0.00	0.06	0.17	1.00	...
manager	0.00	0.71	0.00	0.06	0.65	0.00	0.07	0.00	0.06	...
$'Chang'$	0.05	0.10	0.06	0.06	0.05	0.00	0.10	0.10	0.06	...
$=$	0.05	0.11	0.06	0.00	0.05	0.00	0.50	0.00	0.00	...
department	0.05	0.15	0.15	0.10	0.05	0.0	0.07	0.05	0.10	...

On the level of queries, especially, the application of selection strategies is of advantage. They refine the final recommendation result and, thus, are influential parameters during the evaluation. Example 5.9 demonstrates the application of the different selection strategies described in Chapter 2. Note that, unlike in the definition of the selection strategies for schema matching, the values are only selected regarding the rows in a matrix of query similarity values.

Example 5.9. Below, the first matrix shows the aggregated and combined results of the Name and Type Matcher. The results of the application of the three selection strategies MaxN, MaxDelta, and Threshold are displayed thereafter. The parameters used are $n = 2$, $d = 0.4$, and $t = 0.4$.

The combined results:

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 0.92 & 0.62 & 0.18 & 0.00 \\ 0.62 & 0.97 & 0.00 & 0.50 \\ 0.18 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.50 & 0.00 & 1.00 \end{pmatrix} \end{array}$$

MaxN:

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 0.92 & 0.62 & 0.00 & 0.00 \\ 0.62 & 0.97 & 0.00 & 0.00 \\ 0.18 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.50 & 0.00 & 1.00 \end{pmatrix} \end{array}$$

MaxDelta:

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 0.92 & 0.62 & 0.00 & 0.00 \\ 0.62 & 0.97 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix} \end{array}$$

MaxThreshold:

$$\begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_3 \end{array} \\ \begin{pmatrix} 0.92 & 0.62 & 0.00 & 0.00 \\ 0.62 & 0.97 & 0.00 & 0.50 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.50 & 0.00 & 1.00 \end{pmatrix} \end{array}$$

In general, the aggregation as well as the selection are applicable in all phases of matching (i.e., with match results of the Feature or Schema Matcher), on query-level, too. This is because all matchers deliver similarity values between 0 and 1. Nevertheless, the application of an aggregation strategy only makes sense if several matchers are applied in one phase.

After limiting the set of possible recommendations with the Feature Matcher, applying different schema matchers (i.e., the Name and the Type Matcher) and aggregating their results within the Schema Matcher, and selecting those values above $t = 0.4$, the query recommendation system would provide the following recommendations for $q_{orig} = q_0$:

$$R_f = \{(q_0, 0.92), (q_1, 0.62), (q_2, 0.0)\}$$

5.3. Conclusion

This chapter presented a combined, content-based approach for query recommendation based on query matching that fulfills the requirements specified in Chapter 3. The procedure proposed works on an abstract query representation and, hence, is independent of specific query formats and languages. Thus, the interoperability criterion is met. The required interactivity is achieved by applying a matching algorithm based on index search, which is of logarithmic complexity. In order to produce recommendations of good quality, several schema matching algorithms are applied. The translation of the query into a schema structure has been optimized for a selection of traditional matchers. In general, however, the approach is independent of particular schema matchers and, hence, can integrate different existing approaches. It also can be configured according to various circumstances, especially to custom

domains. Finally, proven aggregation and selection strategies from the domain of schema matching have been incorporated into the query matching framework to enable the parallel execution of matchers and to improve the result.

Nevertheless, to demonstrate its usefulness, the query recommendation system has to be evaluated in an adequate setting. This is covered in the next chapter, which presents details about the empirical evaluation conducted in the course of this study.

6. Evaluation

This chapter describes the evaluation of the query recommendation system developed in this work. For the creation of an evaluation-data set, which presents a major contribution of this thesis, an empirical study has been conducted and is described in the first section. Next, follows an overview of the evaluation-setting, the presentation, and the analysis of the results.

6.1. Empirical Study

With recommender systems and query completion methods, current approaches for query recommendation focus on goals different from remix. For both, obvious evaluation methods exist: for the former, query recommendations are rated successful if they belong to one session; for the latter, recommended completions for a partial query can be compared to the original query. Note that query completion evaluations also use session information [51] or directly ask users to rate the system results [47], in order to include a larger context. Since remix does not rely on session information and only works with complete queries, neither of the existing evaluation methods is appropriate for an automated, large-scale evaluation. Hence, its evaluation requires a set of queries where correct recommendations for a query were marked in advance – based on a general, human perception of query similarity.

This section describes the alternatives for the set of queries and the empirical study conducted to augment them with similarity information. The latter includes the survey design, preparation, implementation, and results.

6.1.1. Data Set Alternatives

The requirements for the evaluation data, given in the following, reflect those of remix described in Chapter 3.

- Similarity values between query pairs (Effectiveness)
- Appropriate variability and complexity (Quality)
- Adequate size (Interactivity)
- Different data sources (Interoperability)
- Different domains (Domain-Independence)

Apart from reports, which are usually not available in larger amounts, there are other query sources which can be considered as data set alternatives. Sets of queries generally can be found in tutorials (e.g., query samples in a tutorial for a query language like SQL), generated by query generation engines with the help of templates (e.g., the engines which come with benchmarks for DBMS like the TPC [15]), or given with the logs of database systems. Table 6.1 gives an overview of how they fulfill the data set requirements for query recommendation.

Nevertheless none of the alternatives fulfills all of the requirements described. The variability and complexity of queries extracted from reports strongly depends on the reports themselves; in general, however, larger sets of different kinds of reports are not available.

Source	**Similarities**	**Variability**	**Complexity**	**Size**	**Interoperability**
Reports		x	x		x
Tutorial		x			
Generation			x	x	x
Query Log		x	x	x	

Table 6.1.: Overview of data set alternatives

Though, interoperability is provided if reports from different tools are regarded. By contrast, queries from tutorials are easier accessible and often include variation. But they usually are neither very complex, nor available in sufficient numbers. Query-generation engines overcome the size objection. But even though they may be complex enough, address different databases, or reflect different domains, they naturally lack the required variability. Query logs of large database systems with different users usually fulfill the variability, complexity, and size criterion. The interoperability and domain-independence of the queries depends on the layout and domains covered by the corresponding databases. Note that the similarity criterion is met by none of the available data sets.

Based on the above analysis, a set of queries extracted from the query log of the Sloan Digital Sky Survey (SDSS) [14] has been used as basis for the data set. The SDSS maintains a very large, open database containing scientific data about the universe. Its query log contains millions of SQL queries issued by different users. Since the SDSS database also contains views for the tables, the data set even partially meets the interoperability requirement. The empirical study conducted in the course of this study to augment queries from the SDSS with similarity values is described in the next section.

6.1.2. Preparation

The original data set was pre-processed in two ways before being used in the survey: In a first step, the set of queries was pre-processed and filtered for reasons of diversification. Second, similarity values were precomputed for obvious cases. Due to the large amount of data the entire processing was done in an automated fashion. The corresponding SQL scripts can be found in the Attachments A.

The first set of queries was chosen arbitrarily (see the query used to retrieve the set in enclosed in the Attachments A) and subsequently processed to filter out specific instances:

1. Query duplicates
2. Queries too long to be grasped by the survey participants in an acceptable amount of time (i.e., queries consisting of more than 350 characters)
3. Erroneous queries

In addition, queries which only differed in values (e.g. 'SELECT * FROM manager WHERE name='Chang" and 'SELECT * FROM manager WHERE name='Chong") were restricted to two examples per occurrence. Note that all nested queries had to be filtered out because they were too long and, thus, not adequate for the survey-setting.

Since all SDSS queries target one database system, several similarity values could be precomputed in advance: all those query pairs that neither query the same table or a corresponding view nor use common attributes should not be similar. Furthermore, every query is very similar to itself. The exclusion of the precomputed similarity values reduced the original 22,500 similarity values to be found to about 9,000.

6.1.3. Design and Implementation

An empirical study validates its findings with a large set of participants. To reach the sufficient number of participants, the 'Query Similarity Survey' was conceived as a Web application. The goal of this survey was to get a similarity value for each pair of queries in the data set through the rating of real persons. Its design and implementation are presented in this section.

In order to get an impression of the SQL knowledge of the participants, every user had to give a rating about his SQL skills in advance. The choice of the rating is given in the following:

- SQL? I know that is something used with databases.
- I learned some SQL at school.
- I did some basic database course at university (or similar).
- I use databases for my work.
- I'm a master of SQL.

In the survey, the users were then presented one source query which they had to compare to three other queries. As similarity rating, one of the following could be chosen:

- Not similar
- Maybe similar
- Similar
- Very similar

A screenshot of the final application can be seen in Figure 6.1. The compositional structure of the survey application is shown as FCM Block Diagram in Figure 6.2. It has been implemented in Java, Java Script, and HTML and is running on a Tomcat Server [2]. The participants comfortably access the application via a Web Browser. Apart from the *QuerySurveyUtils*-component, the application has been implemented independent of the actual data. The latter is processed as text and stored in a MySQL [13] database, where it can be administrated directly.

The independence of specific survey data presents a major benefit of the application and makes it usable as a framework for surveys about similarity of arbitrary objects. With 875 lines of Java code (LOC) it is not too complex to handle and can be reused in future similarity evaluations for remix (e.g., regarding visualization objects). At the time of finishing this work, a PhD student is using the tool for retrieving similarity values for ontological concepts.

6.1.4. Results

This section summarizes details about the survey results and describes the function applied to transfer the user ratings to similarity values.

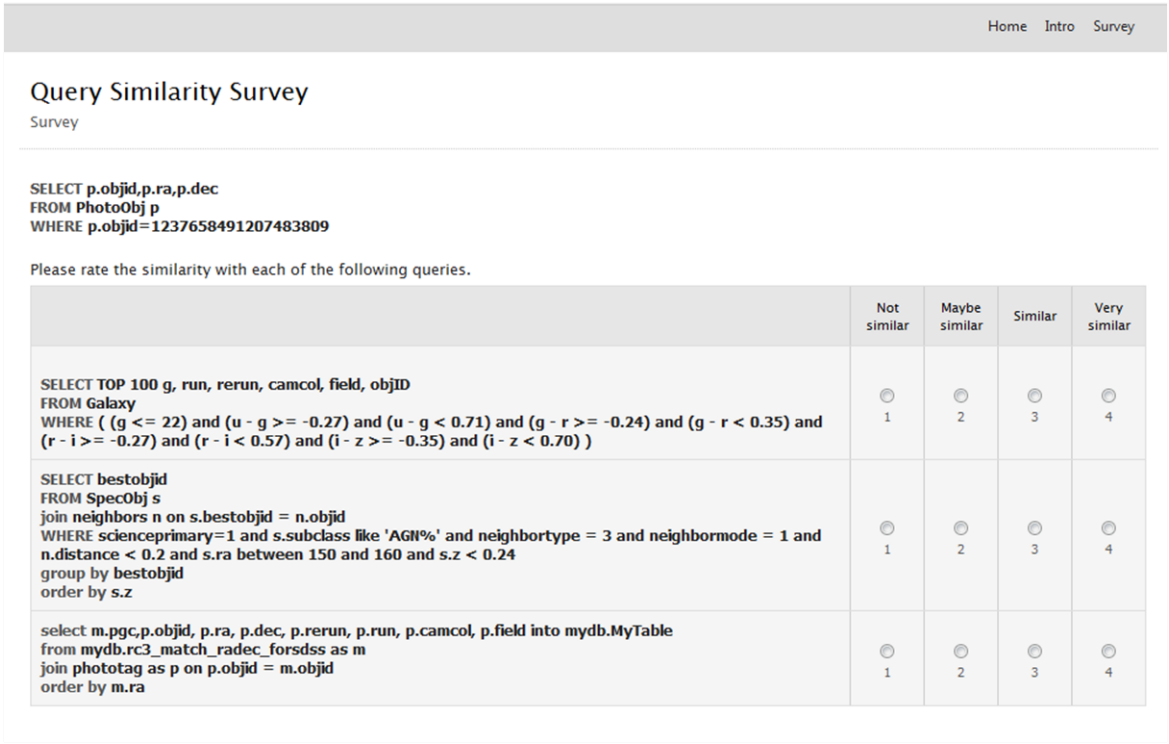

Figure 6.1.: Screenshot of the survey application

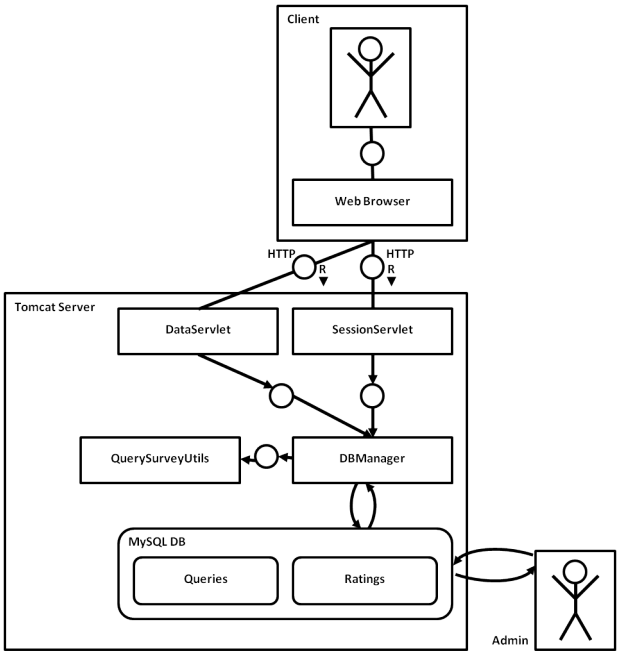

Figure 6.2.: The compositional structure of the survey application

The survey was started by 121 participants. About 50% of them reported to have basic SQL knowledge, 40% had deeper, and 10% only rudimentary knowledge of SQL. On average, the participants rated 82 queries. Conversely, the 8,595 retrieved similarity values were averagely rated by 1.2 people, who in about 70% of the 1,307 cases with multiple votes agreed in their rating. All in all, the study lead to a data set of 150 queries with similarity

values between all of them.

The function $sv : [0..3] \rightarrow [0..1]$ used to compute a similarity value for a pair of queries based on several ratings of survey participants is defined as follows:

$$sv(r) = \begin{cases} 0 & \text{if } r = 0 \\ 1 & \text{if } r = 3 \\ (r + 0.5) * 0.25 & \text{else} \end{cases}$$

The function takes the average rating of the users as input and transforms it into a similarity value in the interval [0..1]. If all users agree on a rating of 'not similar' or 'very similar', the value is 0 or 1, respectively. Otherwise, the function precisely reflects the average rating by selecting the corresponding value in the interval of the rating. For that, the interval [0..1] is split into the intervals [0..0.25[, [0.25..0.5[, [0.5..0.75[, and [0.75..1], which correspond to the four rating possibilities. For example, given an average rating of 1.5 (i.e., between 'maybe similar' and 'similar') for a pair of queries, function sv returns exactly 0.5.

In contrast to evaluations in IR or recommender systems which often only apply a reference set with a binary scale [41], this study considers a more fine-grained data set using a numerical scale. Nevertheless, the ratings of the users often agree and certain values accumulate. For the limited sets of queries considered containing always only a subset of similar queries, however, the scale of values applied in this study offers enough possibilities to evaluate the recommendation system w.r.t. different aspects of similarity computation. In particular, it allows for a consideration of the ranking.

After the detailed description of the retrieval of the test data in this section, an overview of the evaluation is given, next.

6.2. Overview

This section gives a detailed description of the evaluation-setting including specifications of the data sets and the individual experiments conducted. Moreover, it covers the implementation.

6.2.1. Data Sets

Table 6.2 gives details about the data sets used in the evaluation. The first set, SDSS, is the result of the empirical evaluation described above. The second data set consists of queries extracted from an SQL tutorial.

Property	**SDSS**	**SQL Tutorial**
Number of queries	150	43
Recommendations per query >3/>5/>10/>15 (in %)	96/87/52/29	61/49/28/2
Min/Max/Avg query length (word count)	6/82/38	4/42/18
Share of nested queries	0/150	7/43

Table 6.2.: Statistics of the test data

As mentioned above, the SDSS data set contains scientific queries addressing a very large DBMS. The queries were issued by different users and sometimes also generated by tools –

there are, for example, query templates, which only have to be filled by the users. In order to have a variable data set, it contains maximal two occurrences of queries with the same structure only differing in values. The similarity values for the data set were retrieved with the empirical study described in the previous section.
The SQL Tutorial data set consists of queries extracted from an SQL tutorial. Most of these queries are about the business domain. Since the queries are intended to show different concepts of SQL, they contain a very variable structure. In sum, however, the queries are not very complex because they target SQL learners. The tagging of the data set with similarity values was done by the author of this thesis.

Further, note that the queries in both data sets are SQL queries.

6.2.2. Experiment Design

This section describes the goals of the evaluation and gives details about the experiments. The experiments conducted in the course of this study focused on the following two topics:

1. **Configuration and Evaluation of Schema Matchers**
   The applicability of schema matchers for query matching was evaluated in the first part of the evaluation. For that, the configuration of the individual matchers had to be determined, in advance. In addition, different combinations of schema matchers were evaluated regarding their query recommendation performance.

2. **Evaluation of the Combined Matcher**
   The actual performance of the query recommendation system developed in Chapter 5, was showed by regarding the accuracy of its recommendations as well as scalability issues. In particular, it addressed the following issues:

   - It showed that the performance of the Feature and the Schema Matcher, which were presented in Chapter 5, is better than methods provided by state-of-the-art systems (e.g., keyword search).
   - It showed that the combination of the Feature and the Schema Matcher outperforms the individual approaches – in particular, that the scalability issues of schema matching are solved by the filtering and that the recommendation accuracy is not affected considerably.
   - It showed that the extension of the similarity function proposed by Apache Lucene [12], which is described in detail in Chapter 5, is of advantage and computes a more accurate similarity value.

The experiments included all the schema matchers presented in Chapter 2 and considered all $2^9 - 1$ combinations (i.e., apart from the general matching approaches, two more specific matchers, the W-Name and the T-Path Matcher, were included. Both proceed like the respective base implementations but break down the element names into tokens; the comparison is then performed on a more fine-grained level). Further, they considered all 11 combination and selection thresholds in the interval [0,0.1,..,1]. Consequently, 61,831 (i.e., $(2^9 - 1) * 11 * 11$) combinations were evaluated. For the configuration of the matchers, every third query was used considering both data sets (i.e., 50 queries of the SDSS data and 14 of the SQL Tutorial queries).

Then, the final evaluation considered the remaining 129 queries of the two data sets and the query recommendation approaches given in Table 6.3. They are, in the following, often addressed by the names given in the table. The effectiveness of the individual approaches

developed in Chapter 5 was determined by comparing the results they delivered to two baseline approaches, the String and the Text Matcher. In particular the String Matcher, which computes the similarity value by relating the size of the largest common substring of two queries to the one of the longer of them, represents the keyword search applied in state-of-the-art DBMS [50]. The Text Matcher maintains an index and performs text search as it was exemplarily described in the problem introduction in Chapter 3. It calculates the similarity values based on a combination of TF-IDF and the cosine similarity[1].

Name	Description
Combi	The Combined Matcher presented in Chapter 5
Schema	The Schema Matcher proposed in Chapter 5
Feature	The Feature Matcher proposed in Chapter 5
String	A matcher computing a string similarity
Text	A matcher performing text search

Table 6.3.: The matchers considered in the evaluation

The measurements applied for evaluating the quality of query recommendation are also described in Chapter 2. The evaluation results presented are usually the average values for all recommendation tasks in one data set (i.e., naturally, the number of recommendation tasks corresponds to the number of queries in the data set). In addition, the scalability was considered. Therefore, the runtime was measured during the evaluation executing all matchers on a usual office PC using an Intel Core 2 Duo, four gigabyte RAM and a Java 6 (32-bit) environment.

6.2.3. Architecture and Implementation

This section covers implementational issues of both the query recommendation system and its evaluation.

Query Recommendation System

Figure 6.3 gives an overview of the architecture of the system, which has been implemented prototypically in Java as part of this work. It accepts textual queries as input (e.g., strings or CSV files), imports, and parses them, and stores them in an abstract representation in the system. These queries then are the base from which recommendations are provided, if a user queries the system.

A major part of the system consists of the functionality for pre-processing queries, which is grouped in the Query-Package shown on the left of Figure 6.3. The pre-processing includes the import of queries from different sources (e.g., custom formats, text files, or databases), the parsing of queries in different query languages, and the translation into an internal format. EMF [7] is used for generating parsers for different query languages based on a model together with a detailed textual description in concrete syntax.

[1] The cosine similarity is commonly used in the Vector Space Model of IR to calculate the distance between two vectors. Given two documents d_1 and d_2, for which the similarity is to be determined, the cosine similarity is defined as $d_{cosine} = \frac{\vec{V}(d_1) * \vec{V}(d_2)}{|\vec{V}(d_1)| * |\vec{V}(d_2)|}$ [20].

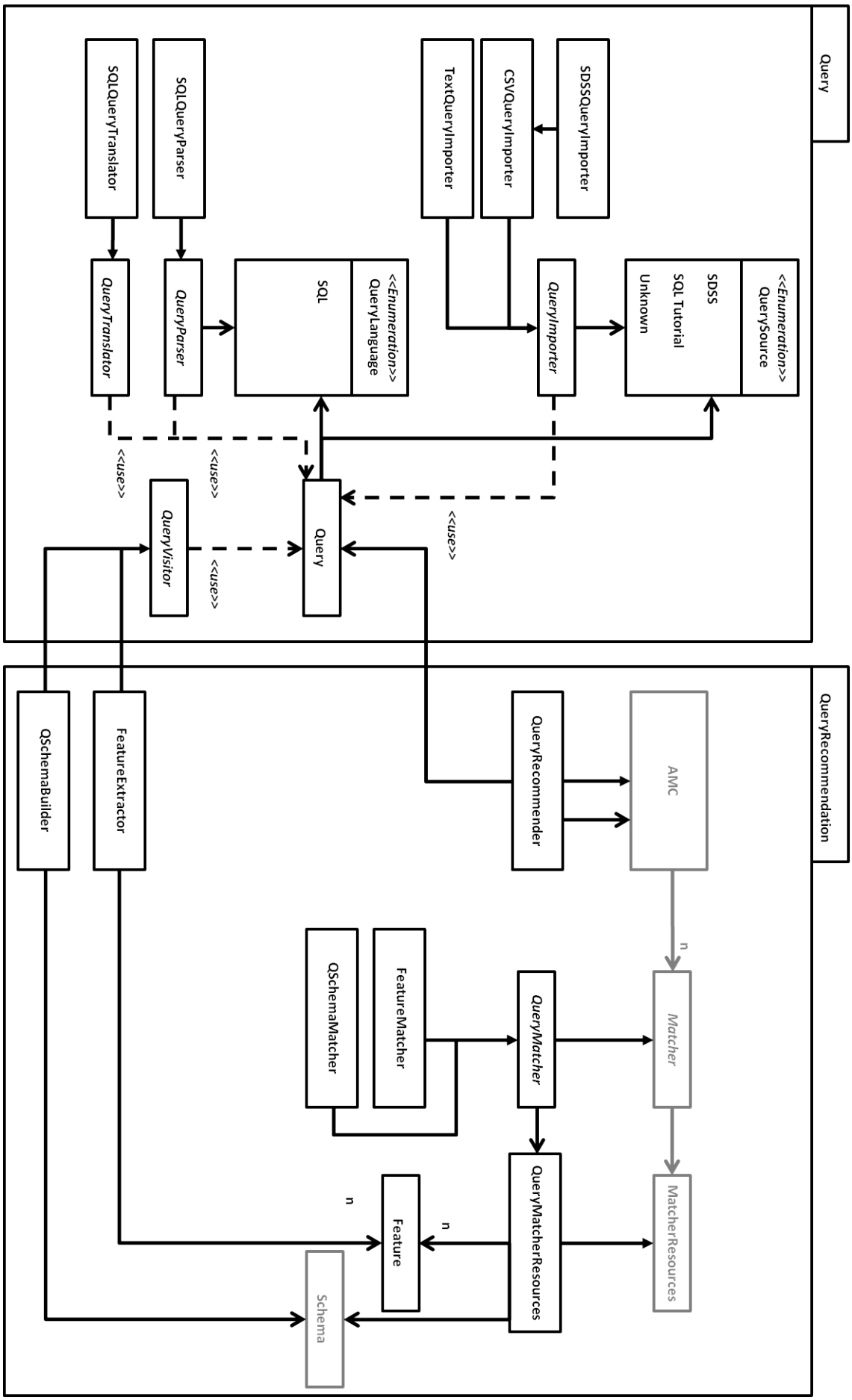

Figure 6.3.: The main parts of the architecture of the system

Furthermore, it allows the generation of the code for the internal query format based on a model. This has the advantage that the model is well-defined and open for adaptation and extension. The generated code, in total, comprises about 30,000 LOC.

During pre-processing, also, specific information necessary for matching has to be extracted and made explicit (e.g., properties of the queries like ids and names of data sources queried). For that reason, an abstract Visitor class (shown center-bottom in Figure 6.3) going through the abstract syntax tree of a query has been developed. Thereby the extensibility of the query format has been taken into account by using reflection (e.g., new operators in the abstract query model would automatically be visited). Concrete implementations of the visitor serve to extract the features or build the schemas, for example.

The recommendation framework, pictured on the left of Figure 6.3, has been built around the AMC, the schema matching system available. For that a Wrapper has been developed that accepts and matches sets of queries instead of two schemas. The adaptations include main components of the AMC and were often done through inheritance (e.g., the establishment of a query matcher hierarchy on top of that of the schema matchers). AMC-components are drawn grey in the picture of the architecture.

The new system strongly reflects the architecture of the AMC. In particular, new query matchers can be directly registered with the recommendation framework, such allowing for easy extension. Nevertheless, the main advantage of the connection to the AMC is that the efficiently implemented schema matching infrastructure can be reused. Next to the efficient execution of several matchers in parallel (e.g., supported by the use of threading and caching), the functionality provided for aggregation and selection represents a special advantage.

Through the AMC, there is also a number of existing schema matchers, which serve as a basis for new matchers tailored to the query structure. The index-based processing of the feature matcher was implemented using the Apache Lucene Search Engine Library [1]. Next to the logarithmic search complexity, it provides customizable methods for the calculation of similarity values.

Evaluation Framework

In order to enable a large-scale evaluation, the entire evaluation has been implemented in SQL stored procedures and was executed directly on a MySQL database. The main advantages are the direct connection to the data set, the results of the empirical study; the performance gains (e.g., especially, the recording and reuse of intermediate results while evaluating schema matcher combinations); and the availability of all data in a format suitable for further analysis. For an exhaustive evaluation, large parts of the schema matching functionality (e.g., the aggregation and selection of the results of individual schema matchers as well as the combination of the element-level results) have been implemented in SQL, too (see the corresponding SQL scripts in the Attachments A).

6.2.4. Threats of Validity

There are two main threats for validity which have to be considered with this evaluation. First, the correctness of the implementation has to be regarded. Second, a more specific issue concerning the preciseness of the calculation needs to be addressed.

In order to ensure the intended operation of the system different tests were performed. For the pre-processing functionality for the queries implemented in Java, tests were defined using JUnit [11]. Moreover, the implementation of the evaluation system has been validated against the original evaluation functionality of the AMC. For that, several results of the matching system were evaluated with both systems to compare the evaluations – by regarding random samples. Similarly, the SQL implementation of the schema matching functionality was tested by comparing its results to those of the AMC.

Further, the similarity calculation performed by Apache Lucene, which is used in the implementation of the Feature Matcher performs a byte encoding/decoding of certain factors, which leads to a loss in precision [12]. Since the over-all ranking is maintained, this does not influence the Combined Matcher, which only applies the values for filtering. However, the results of the Feature Matcher are sometimes – although, only marginally – influenced.

This section presented in detail the test data, experiment design, and architecture of the system and covered issues of validity. The results of the evaluation of the system are given in the next section.

6.3. Results

This section presents the results of the evaluation. The first section covers the configuration and evaluation of the individual schema matchers as well as their combination. Next, the quality of the Combined Matcher conceived in Chapter 5 is evaluated w.r.t. different aspects of query recommendation accuracy and in comparison to different approaches. In particular, the usage accuracy and the scalability of the combined approach are examined closer. Lastly, the rating and ranking accuracy, two other aspects of query recommendation are regarded.

6.3.1. Configuration and Evaluation of Schema Matchers

In this section, the applicability of schema matchers for query matching and recommendation is evaluated. For that, the best configuration for the individual matchers is derived, first. It includes the best combination and selection threshold for each matcher. After a quality comparison of the individual matchers, the best combination of matchers is presented, which is applied within the Schema Matcher. Since the over-all goal of this work is query recommendation, pr@5 is chosen as the most important measure for the evaluation[2][3]. Hence, the parameter selection described in this section targets the optimization of this value.

For the computation of the result of a schema matcher, the match results on element level need to be coerced into one value. Prior to the application of the schema similarity function, *ssim*, the real element correspondencies are selected using the combination threshold, as described in Chapter 5. Since the then computed value of the *ssim*-function determines the similarity value of the corresponding queries, the MAE, which captures the preciseness of the similarity value serves best to determine the combination threshold to be chosen. Recall that the MAE captures the error (i.e., it describes the average deviation of the computed similarity

[2]Note that the presentation of three recommendations to a user might be a more reasonable concept, in reality. Then, pr@3 should be optimized. However, this study considers pr@5 because the data sets always include the original query and the SDSS data, in addition, contains several structural duplicates. In this regard, pr@5 seems a more reasonable choice.

[3]Since pr@k is regarded within the evaluation, it has to be noticed that in some cases (i.e., if the reference set contains less than k recommendations) a pr@k of 1 cannot be achieved.

values from the reference values) and, hence, should be minimized.

Figure 6.4 exemplarily shows the development of MAE@5 and MAE@10 for different combination thresholds for the Name Matcher. For both measures, there is a strong degression until a combination threshold of 0.7. After that point, the curves decline more slightly. Note that pr@5 has its peak at the combination threshold of 0.7 (not shown here). Therefore, it is selected as aggregation threshold for the Name Matcher.

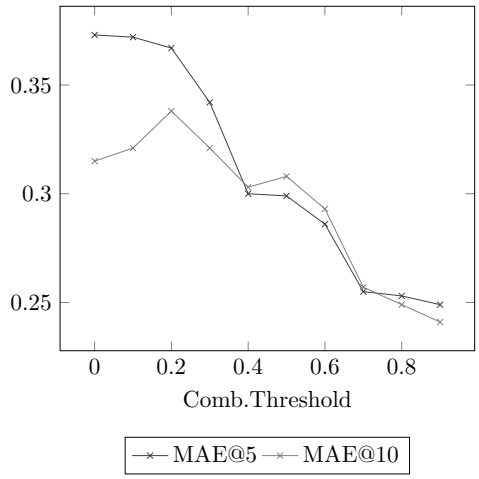

Figure 6.4.: The dependence of the MAE from the combination threshold for the Name Matcher

Similarly, the best combination thresholds have been determined for all other schema matchers and are shown in Table 6.4. Nearly all values are very high (i.e., greater or equal 0.7) apart from those for the Type and Leaf Matcher. This is in accordance with the assumption of the schema similarity function, which assumes element-pairs with similarity values greater than 0 to be real correspondencies.

Matcher	**Comb.Threshold**	**Threshold**
Name	0.7	0.4
W-Name	0.7	0.4
Path	0.9	0.2
Sibling	0.7	0.2
Type	0.0	0.7
Leaf	0.5	0.1
Parent	0.7	0.5
Children	0.8	0.0
T-Path	0.9	0.3

Table 6.4.: Configuration parameters for the schema matchers

In the right column, Table 6.4 gives the best selection thresholds for the individual schema matchers. In contrast to the previously found combination thresholds, here, the Type Matcher has the highest threshold with a value of 0.7. This is to be traced back to its processing. Since

data types mostly either match or not the combination threshold of 0 is chosen because a higher threshold would not change the result.

The dependence of different measures regarding the threshold can be seen in Figure 6.5, again, for the Name Matcher. The two charts also show that a threshold-selection focusing on the measures precision, recall, and F-Measure (i.e., a consideration of the whole set of recommendations) would lead to a different best-threshold of about 0.7. This indicates that the precision for the current best-configuration decreases, if a larger set of recommendations is regarded.

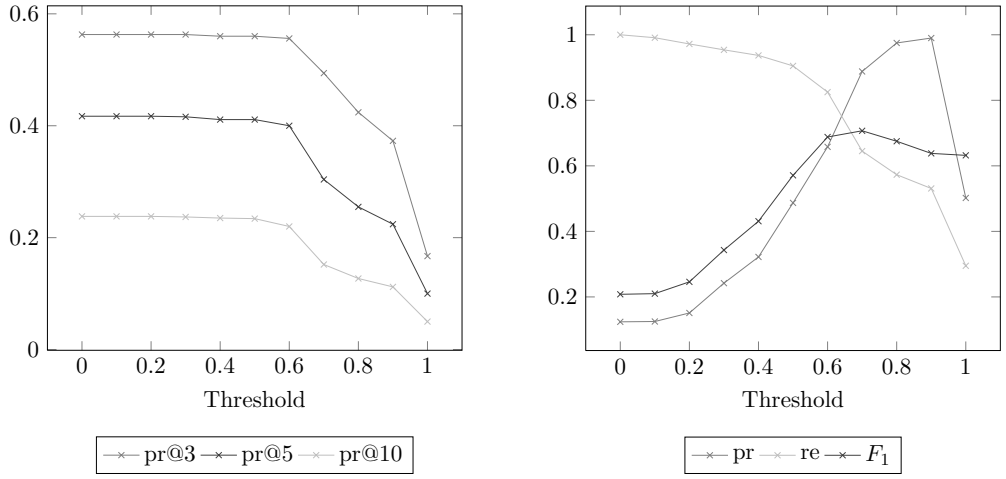

Figure 6.5.: The influence of the Threshold selection for the Name Matcher using different thresholds

Figure 6.6 shows the final evaluation of the schema matchers configured with the parameters specified in Table 6.4. In general, there is no significant difference between the individual matchers. Especially, with the Sibling and T-Path Matcher, the pr@5 is at 0.6; with the other matchers, it is between 0.5 and 0.6.

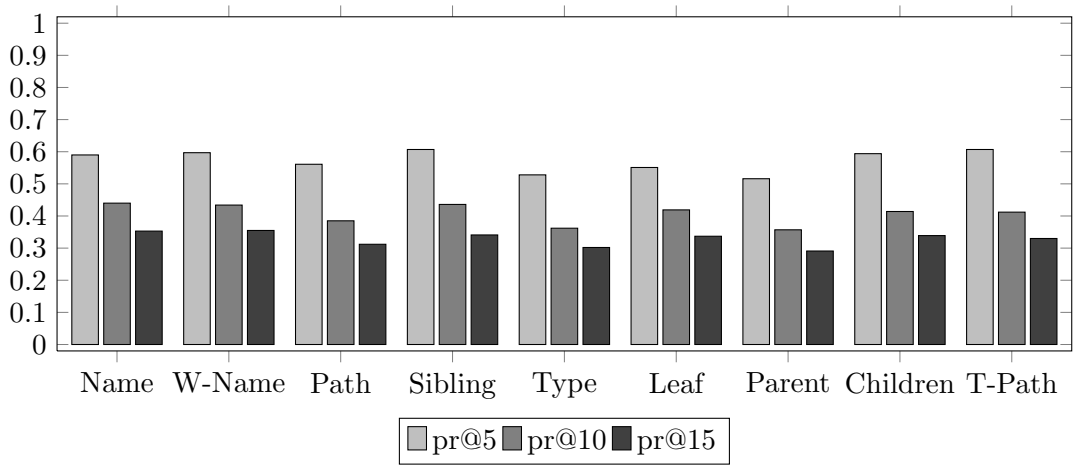

Figure 6.6.: The evaluation of the individual schema matchers regarding their usage accuracy

The good performance of the Sibling Matcher shows that its processing fits well with the structure of the queries. Since most of the queries are not nested, their schemas usually consist

of one operator level and one attribute/value level. Both levels are very homogeneous such that the Sibling Matcher can achieve good results. As it is the case with pr@10 and pr@15, which have very similar values for the different matchers, all other measurements not shown here (e.g., precision and recall) do not differ significantly.

Regarding the combination of schema matchers, the combination of the W-Name, Leaf, and Children Matcher turned out to achieve the best quality – having aggregated with the Average Strategy and applied a selection threshold of 0.4. We considered all 511 possible combinations of matchers and the best 30 combinations only included matchers of these three kinds together with the Name Matcher. The differences in the evaluation results for these combinations were sometimes only marginal. Compared to other combinations (i.e., including other matchers), however, the drop in quality made up about 5% for pr@5 and about 10% for F-Measure, for example.

Note that the Leaf and Children Matcher both focus on comparing the attributes, which are the parts most distinctive with different queries. The T-Path Matcher, in addition, includes information about the query structure in its comparison. Thus, the best combination considers two important kinds of information contained in the queries.

6.3.2. Evaluation of the Combined Matcher

This section covers the evaluation of the Combined Matcher presented in Chapter 5. A preliminary evaluation turned out that it achieves best results with a filter-parameter of $n = 20$. That means the Filter Matcher ranks the queries after calculating preliminary similarity values and forwards the best 20 queries to the Schema Matcher. The latter recalculates the similarity values for the 20 queries and the similarity values of the remaining queries, which are not considered by the Schema Matcher, are set to 0.

Usage Accuracy

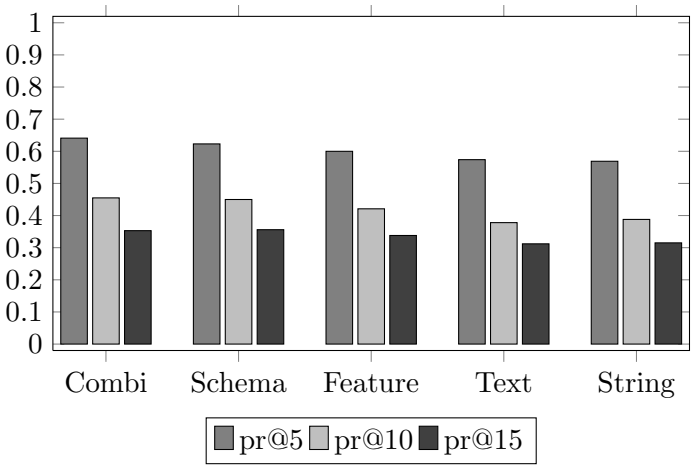

Figure 6.7.: The evaluation of the query matchers regarding their usage accuracy

It turned out that all approaches considered have a very similar performance considering the top k recommendations. An overview of pr@5, pr@10, and pr@15 is given in Figure 6.7. Though the similarity of the results, only the Schema Matcher and the Combined Matcher achieve a pr@k greater than 0.6. The Feature Matcher shows a better quality than the baseline

approaches, but the differences seem to be marginal; the values all are between 0.5 and 0.6. This induces that with all matching approaches, about 50% of the top five recommendations are real recommendations. Note that the values also resemble the results achieved by the individual schema matchers, which were presented in Figure 6.6.

Nevertheless, the consideration of precision, recall and F-Measure shows differences in the matching performance of the different matchers. Although all of the latter find the real recommendations, which is indicated by the high recall, their over-all matching performance differs strongly. With F-Measure values of 0.38, 0.47, and 0.56 for the Feature, Schema, and Combined Matcher, respectively, the increase in the value makes up about 10%. With a precision of 0.47, only the Combined Matcher is near 0.5. This indicates that it is the only matcher where at least half of all recommendations determined are useful.

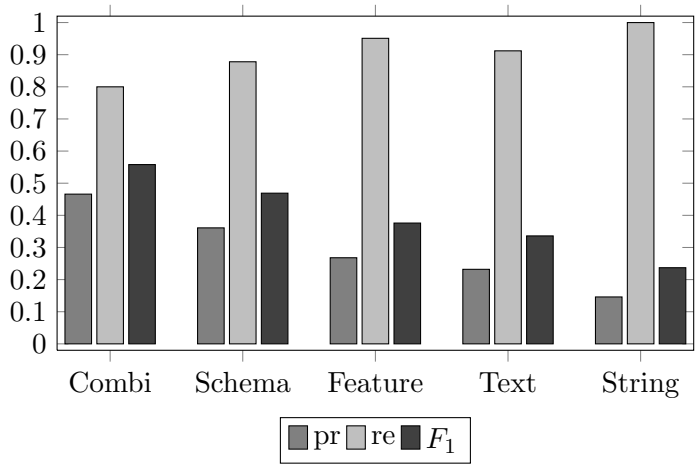

Figure 6.8.: The evaluation of the query matchers regarding their over-all matching performance

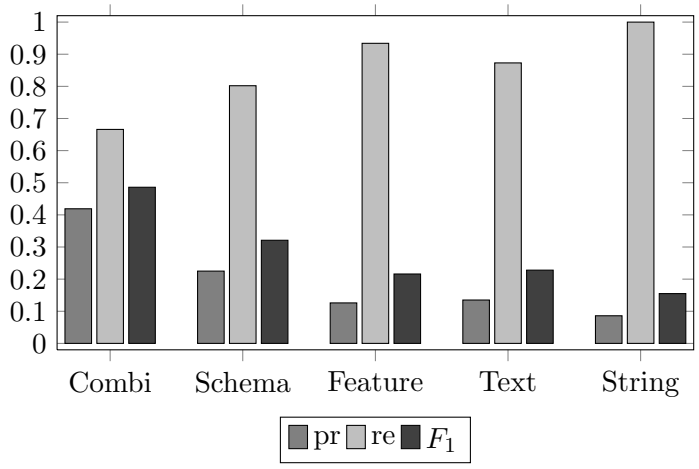

Figure 6.9.: The matching performance of the query matchers regarding the SDSS data set

If only the SDSS data set is considered, the advantage of the Combined Matcher is even more evident. In comparison to the average F-Measure shown in Figure 6.9, for example, all values drop by 25-50%. Only with the Combined Matcher there is a smaller decrease (of less

than 10%); hence, its recommendation quality proves to be much more constant. The good performance of the combined approach in contrast to the individual matchers indicates that the filtering succeeds in filtering out queries that would be falsely matched by the schema matching system. Moreover, the schema matching overcomes the very generous strategy of the Feature Matcher, which achieves a high recall but a very low precision. It refines the rough similarity calculation of the Feature Matcher and leads to a significant increase in precision for the Combined Matcher.

Scalability

Next to the recommendation accuracy, the scalability of the recommendation system was covered by the evaluation. The results are shown in Figure 6.10, which gives an overview of the time the systems need for processing in dependence of the number of queries in the systems. The data confirms the specifications of the complexity of the approaches, stated previously.

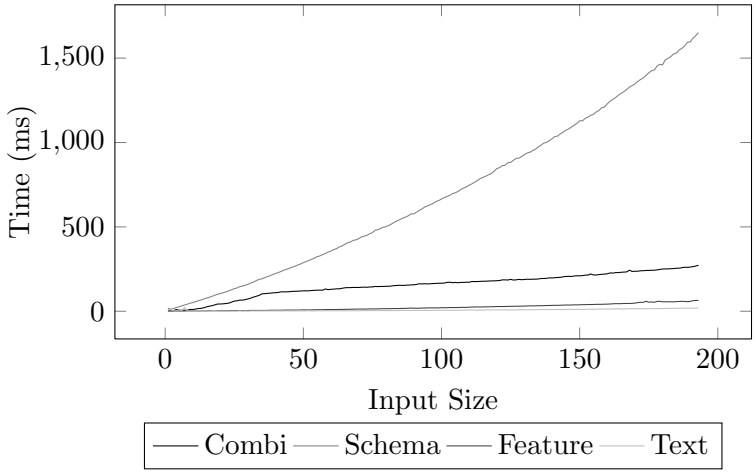

Figure 6.10.: The evaluation of the query matchers regarding their scalability

The curves of the two baseline approaches both develop rather steadily with only a slight increase with growing input. Although it has to maintain the terms, the Text Matcher requires less time than the String Matcher (not shown here). Using an index it achieves logarithmic complexity.

Similarly using an index, the Feature Matcher shows the same development. However, since it is based on the abstract representation of the queries, the pre-processing required by the approach leads to a larger time in processing. Also the quadratic complexity of the schema matchers is reflected by the time they need for processing, which even increases along the quadratic parable. Finally, the Combined Matcher shows a logarithmic curve, too. Since the filter-parameter, n, sets a bound for schema-matching, its complexity is based on the indexing of the filtering.

Rating and Ranking Accuracy

Figure 6.11 shows the MAE of the Feature Matcher and the original version of similarity computation based on Lucene and indexing all Features. The extensions clearly benefit

the computation since all MAE values are lower with applying them. In particular, the MAE@5 decreases by about 30% of its value with the original computation. Further is to be remarked that, for different MAE, the values stay constantly at about 0.2 with the extended computation, whereas the original version shows variation in the value. This is because the similarity calculation of the latter includes all features and, hence, behaves different and becomes inaccurate if more queries are considered. Note that regarding other measurements, the two approaches differ only slightly. This is because the extended similarity calculation maintains the over-all ranking of the feature-based approach, but only has a refined choice of values.

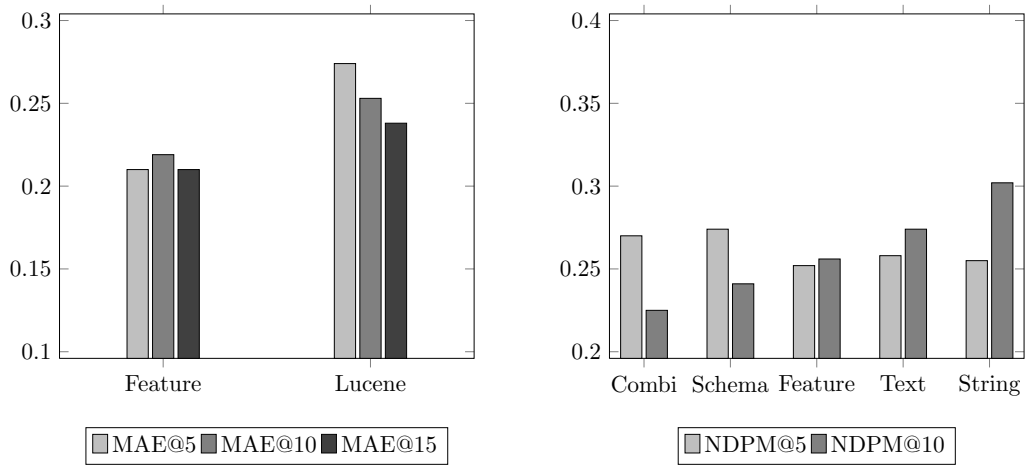

Figure 6.11.: The evaluation of the query matchers regarding their rating (left) and ranking accuracy (right)

In contrast to the MAE, which measures the absolute difference in values, the NDPM serves better to regard the effectiveness of the similarity computation. In essence, it shows if the reference ranking is maintained by counting pairs of queries which are ranked invertedly compared to the reference ranking. Hence, a small NDPM value implies a better reflection of the reference ranking.

Figure 6.11(right), again, shows that the Combined Matcher performs better if a larger set of queries is considered – now, for the SQL Tutorial data. A fact reproduced even more clearly with a $k > 10$ (not shown here). Consequently, the similarity calculation of the Combined Matcher does not only succeed in averagely reflecting the similarity values of the reference set, which is shown by the low MAE, but also maintains the relation between individual similarity values in the data set.

Note that details for the MAE values are not given here, because all ranging between 0.15 and 0.3, they do not differ strongly for the different matchers. The only significant remark to be made in that context is that especially the text seach based on TF-IDF always achieves an MAE of below 0.2, and, hence, performs better than all other approaches having values between 0.2 and 0.3.

The results presented in this section raise several issues regarding the different matching approaches. For that reason, the results are summarized and discussed, next.

6.4. Summary and Discussion

This section gives a summary of the results presented above and discusses them, subsequently. Thereby, both the evaluation of the schema matchers as well as the performance of the Combined Approach is considered.

6.4.1. Summary

The configuration and evaluation of the schema matchers included the following items:

- The configuration of the individual schema matchers by finding the best combination and selection threshold for each matcher
- The evaluation of the individual schema matchers regarding the query recommendation accuracy
- The finding of the best combination of schema matchers

The evaluation shows that the results of the traditional schema matchers do not differ strongly and are very similar for the two data sets of rather different domains. Moreover, the similarity in performance is shown by regarding different measures for the usage, rating, and ranking accuracy. Nevertheless, based on an exhaustive evaluation of all possible combinations, a specific selection of matchers was proposed as best combination. It consists of the W-Name, Leaf, and Children Matcher. Hence, for matching, this combination considers both the structure of the queries as well as the attributes and values contained in it.

The evaluation of the Combined Matcher addressed its query recommendation performance by considering several points:

- The comparison of the two new approaches, the Feature and Schema Matcher, to the baseline approaches
- The evaluation of the Combined Matcher regarding usage accuracy and scalability
- The consideration of the rating and ranking accuracy

According to the evaluation of the usage accuracy, the matchers considered achieve comparable results regarding the goal of providing a set of query recommendations. In contrast, the consideration of a larger set of queries, which amounts to the comparison of the real query matching performance clearly shows the dominance of the Feature and Schema Matcher over the baseline approaches and that the Combined Matcher outperforms all others. The usage accuracy thereby was determined based on the three measures precision, recall, and F-Measure, which are commonly used in the matching domain. The scalability evaluation shows that the processing time of the Combined Matcher develops logarithmic in dependence of the number of queries in the system. This is mainly due to the bound on the queries to be compared by the Schema Matcher, which is set with the filter parameter $n = 20$.

Lastly, the rating and ranking accuracy, two important points to be considered with recommendation, were regarded. The MAE of the Combined Approach does not provide any improvement over this of the baseline approaches. However, it is rather low, in general. Further, the ranking defined by the reference set is maintained in most cases, which is shown by the low NDPM values. Especially, the Combined Matcher succeeds in reflecting the reference ranking w.r.t. a larger set of then usually more heterogeneous queries.

6.4.2. Discussion

Although the evaluation is based on an empirically-retrieved data set and contains an exhaustive number of test cases, it has to be regarded critically w.r.t. the test data. Mainly, for two reasons: First, the data sets do not fulfill all requirements specified in the beginning of this chapter. Second, the actual goal of remix, the recommendation of queries from reports, is not specifically considered.

The data sets were shown to fulfill the major criteria in order to reflect the requirements of remix. Specifically, the queries in the SDSS data set are of adequate complexity, show variability, and are available in large numbers. However, the complexity of the queries cannot be exploited entirely, since queries too complex to be grasped by the participants were filtered out for the empirical study. Further, the interoperability required by remix – especially, the acceptance of queries in different query languages – is not covered by the data sets, which contain only SQL queries. Also, the similarity values retrieved in the empirical study are assumed to be wrong in some cases.

On the other hand, remix is introduced as reporting tool in the introduction. However, queries extracted of real reports are not contained in the test data. But being often tool-generated and very long or addressing special BI data, these queries are rather specific. Hence, the consideration of general SQL queries is to be regarded critically considering the reporting context of remix.

Nevertheless, the Combined Matcher has been designed w.r.t. the criteria defined in Chapter 3. It includes an abstract representation to meet the interoperability requirement and applies schema matching, which should produce even better results with multiply nested queries. Hence, it can be concluded that the evaluation presented in this chapter showed the applicability and advantages of the combined approach for query matching, conceived in Chapter 5. However, the specific performance in a reporting environment should be determined with the additional, more specific data sets.

7. Conclusion And Future Work

This chapter summarizes the presented work. In addition, it outlines extensions of the evaluation and discusses improvements of the combined matching approach as directions of future research.

7.1. Conclusion

Over time, a reporting system is used to create a considerable amount of reports, which contain queries. If this information is collected, a valuable knowledge base originates. To date, however, this knowledge is rarely used further. Therefore, this work studied content-based query recommendation, the recommendation of existing queries based on a calculation of similarities between the queries.

For this purpose, the problem of content-based query recommendation was analyzed in detail, which, in large parts, amounted to a study of query matching approaches. A classification for the latter was developed in the preparation of the subject, based on the basic classification of schema matching proposed by Rahm and Bernstein [70].

As a major contribution, this thesis presented a content-based approach of query recommendation based on a combination of two different matching techniques in the Combined Matcher, similarity search using semantic features and schema matching. The system developed fulfills the requirements of the target reporting system, remix, given in Chaper 3. Due to its independence of specific query formats and languages it can be easily integrated in remix; the required interoperability provided by the Combined Matcher is a feature not supported by existing systems. Also, its processing time was shown to be kept in reasonable limits w.r.t. the amount of queries in the system, in the evaluation – a logarithmic dependence is achieved by limiting the number of queries to be processed by the schema matchers by filtering them in advance. Regarding the over-all goal, the recommendation of a certain number of queries, the restriction of the set of recommendations is necessary, anyway.

Further, the evaluation showed that the quality of the Combined Matcher outperforms that of baseline approaches like string comparison and text search and provides better results than the feature-based matching and schema matchers applied individually. Especially, regarding one of the most important criteria, the usage accuracy. The values of pr@5 and F-measure are about 10-20% better than those of the baseline matchers, which was demonstrated with test data from different domains.

The empirical evaluation conducted in this study, represents the second main contribution of this work. Since adequate test data (i.e., a set of real-world queries where the similarity of the queries has been rated by real persons) for query recommendation did not exist prior to this study [61], an empiricial study was conducted to retrieve similarity ratings by real persons. This study led to a test data set of 150 queries with detailed values (i.e., in the interval of [0..1]) for the similarity of all 22,500 query-pairs.

In the course of the evaluation, particular interest was laid to the applicability of schema matching for query recommendation. Although it was shown that the potential of several individual schema matchers could not be exploited specifically in most cases, their combination turned out to deliver better recommendations in comparison to the baseline approaches. In particular, if the number recommendations to be provided is larger. Since this balance of correctness and completeness is exactly that what makes a good matching approach, in general, schema matching represents a useful technique for matching queries. Moreover, its scalability issues were solved in this thesis with the proposal of the Combined Matcher.

Nevertheless, a critical consideration of the evaluation results pointed out that the focus of remix, which is on reporting, should be considered more specifically. Next to that, there are various other directions left for future work.

7.2. Future Work

The study of query matching and the development of the Combined Matcher revealed certain fields which would be worth further study. Refinements on conceptual and implementational level could lead to more precise recommendations and a more specific evaluation would reveal more details about the application benefits. These issues are covered in this section as directions of future research.

Reporting The Combined Matcher conceived in this thesis should be evaluated with queries extracted from reports. These queries are usually nested and considerably longer than the queries considered in this study, which have a maximal length of 82 words. An example is shown in Listing 7.2. Further, report queries target BI data and are often multidimensional. Such a data set also needs similarity values for the different query pairs. In this context, a study as the one conducted in this work is not appropriate since the queries are too long and complex to be grasped by participants with basic SQL knowledge in a reasonable amount of time. Hence, a new concept for the retrieval of similarity values has to be developed, in addition.

Multidimensional Queries With the focus on collaborative approaches instead of content-based techniques, the processing of recommender systems differs from query matching, in general. However, having similar goals, the concepts studied in the field of recommender systems should be examined closer w.r.t. their applicability for query matching. In particular, there are several recommender systems targeting multidimensional queries, which have not been considered in this work. In that context, The rather basic content-based methods applied in recommender systems (e.g., methods based on string similarity [61]) could provide a starting point for matching multidimensional queries.

Test Data Generation The construction of generation engines represents one approach to come to a set of test data. However, a valid data set needs to be based on real world queries and real world ratings. Therefore, algorithms would need to be conceived that include such information. Given an empirically retrieved subset of the necessary similarity values as starting point, machine learning could be applied, for example, to generate the remaining values. Nevertheless, the validity of such a data set would need to be evaluated separately (i.e., if the transfer of similarity values based on predefined characteristics, which are exploited by the machine learners, complies with the perception of similarity of real persons).

Clustering The equality-based comparison of the Feature Matcher provides an effective filter; but in several cases it might turn out to be too restrictive; especially, in a heterogeneous environment. For that, other comparison possibilities should be evaluated. An approach based on clustering would be one possibility. The clustering of features with very similar values (e.g., attributes sales and totalsales) could establish a similarity between queries, which is not recognized by the current approach during filtering. However, the major open issue in this context is how to integrate the clustering with the index-based comparison.

Schema Matchers This work evaluated several traditional schema matchers as well as combinations of them w.r.t. query matching. Nevertheless, there are several issues to be regarded closer including the configuration of the matchers (e.g., one could evaluate further aggregation strategies) as well as their selection (e.g., other kinds of matchers like, for example, matchers based on machine learning could be considered). Since the application of schema matching turned out to be beneficial for query matching, these points should be considered in future work.

Query Semantics The classification proposed in this work mentioned query semantics as important source of information. In the Combined Matcher, however, the semantics are only applied for creating an abstract representation. Therefore, the impact of applying semantic methods should be examined closer. There is a lot of theoretic work on the similarity of queries regarding their semantics (e.g., the problem if two queries have overlapping result sets). The application of semantic methods might be of particular advantage if the ranking of recommendations is to be refined or the queries available for recommendation are all syntactically very similar.

```
SELECT
  (SUM(dbo.V_ADRM_IMS_FCT.ADRM_FORECAST_FC_CURR))/1000,
  dbo.V_ADRM_IMS_FCT.CAL_YEAR,
  dbo.V_ADRM_IMS_FCT.CAL_QTR,
  dbo.V_ADRM_IMS_FCT.CUSTOMER_DESC,
  dbo.MASTER_CODE_DIM.master_code_desc,
  (SUM(dbo.V_ADRM_IMS_FCT.ESTIMATED_IN_FC_CURR))/1000,
  (SUM(dbo.V_ADRM_IMS_FCT.COMMITTED_FC_CURR))/1000,
  (SUM(dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR))/1000,
  (SUM(dbo.V_ADRM_IMS_FCT.UPSIDE_FC_CURR))/1000,
  Closing_DATE_DIM.day_date,
  dbo.REGION_DIM.Global_country_desc,
  CASE WHEN (dbo.REGION_DIM.level_02_code = 'NA')THEN 'NA' WHEN (dbo.
     REGION_DIM.level_02_code = 'LA')THEN 'LA' ELSE dbo.REGION_DIM.
     level_03_code END,
  CASE
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL AND dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR IS NULL THEN 0
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL THEN dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR/1000
```

To be continued on the next page.

```
 ELSE dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR/1000 END,
  SUM(dbo.V_ADRM_IMS_FCT.COMMITTED_FC_CURR),
  SUM(dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR)
FROM
  dbo.REGION_DIM INNER JOIN dbo.V_ADRM_IMS_FCT ON (dbo.V_ADRM_IMS_FCT.
      REGION_KEY=dbo.REGION_DIM.region_key)
   INNER JOIN dbo.DATE_DIM  Closing_DATE_DIM ON (dbo.V_ADRM_IMS_FCT.
      CLOSING_DATE_KEY=Closing_DATE_DIM.date_key)
   INNER JOIN dbo.ACCOUNT_DIM ON (dbo.V_ADRM_IMS_FCT.ACCOUNT_KEY=dbo.
      ACCOUNT_DIM.account_key)
   INNER JOIN dbo.MASTER_CODE_DIM ON (dbo.V_ADRM_IMS_FCT.MASTERCODE_KEY=
      dbo.MASTER_CODE_DIM.master_code_key)
WHERE
  (
   dbo.V_ADRM_IMS_FCT.CAL_YEAR  IN  ( 2011  )
   AND
   dbo.V_ADRM_IMS_FCT.CAL_QTR  IN  ( 2  )
   AND
   CASE WHEN  (dbo.REGION_DIM.level_02_code = 'NA')THEN 'NA' WHEN (dbo.
      REGION_DIM.level_02_code = 'LA')THEN 'LA' ELSE dbo.REGION_DIM.
      level_03_code END  IN  ( 'AP','DACH','EMEA','JP','LA','NA'  )
   AND
   dbo.ACCOUNT_DIM.account_desc  IN  ( 'Software revenue'  )
   AND
   CASE
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL AND dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR IS NULL THEN 0
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL THEN dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR/1000
 ELSE dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR/1000 END  >=  1
  )
GROUP BY
  dbo.V_ADRM_IMS_FCT.CAL_YEAR,
  dbo.V_ADRM_IMS_FCT.CAL_QTR,
  dbo.V_ADRM_IMS_FCT.CUSTOMER_DESC,
  dbo.MASTER_CODE_DIM.master_code_desc,
  Closing_DATE_DIM.day_date,
  dbo.REGION_DIM.Global_country_desc,
  CASE WHEN  (dbo.REGION_DIM.level_02_code = 'NA')THEN 'NA' WHEN (dbo.
      REGION_DIM.level_02_code = 'LA')THEN 'LA' ELSE dbo.REGION_DIM.
      level_03_code END,
  CASE
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL AND dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR IS NULL THEN 0
WHEN
dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR IS NULL THEN dbo.V_ADRM_IMS_FCT.
    COMMITTED_FC_CURR/1000
 ELSE dbo.V_ADRM_IMS_FCT.PROBABLE_FC_CURR/1000 END,
  dbo.V_ADRM_IMS_FCT.OPPORTUNITY_ID,
  dbo.ACCOUNT_DIM.account_desc ;
```

Code Listing 7.1: An example report query

A. Appendix

Query for Test Data

```
SELECT TOP 1000 MIN(seq) AS seq, statement
FROM SqlLog
WHERE error=0
AND dbname='BESTDR8'
AND yy=datepart(yy,getdate()) and mm=(datepart(mm,getdate()) 3)
AND statement not like '%CREATE%'
AND statement not like '%insert into%'
AND statement not like '%ALTER TABLE%'
AND statement not like '%exec%'
AND statement not like '%drop%'
AND statement not like '%set @%'
AND statement not like '%declare %'
AND statement not like '%cross apply %'
AND clientIP not in
(SELECT clientIP
FROM SqlLog
WHERE error=0
AND yy=datepart(yy,getdate()) and mm=(datepart(mm,getdate()) 3)
GROUP BY clientIP
HAVING count(*)>100)
GROUP BY statement;
```

Code Listing A.1: The query used to retrieve the queries for the SDSS data set (extracted June 6, 2012)

SQL Scripts of the Evaluation

```
#aggregate.sql
#aggregates the results of the m individual schema matchers
#(contained in table `resultdb`.`sm_matcher`)
#for all (2^n) 1 possible combinations

#size of the result of every matcher
SET @rcount = (SELECT SUM(sm_result_count(`id`)) FROM `param_src`);
#num of aggs considered
SET @acount = (SELECT COUNT(*) FROM `param_agg`);

delimiter
CREATE PROCEDURE init_view_sm_matcher()
BEGIN
CREATE OR REPLACE VIEW `sm_matcher` AS
SELECT id 1 AS id,`id` AS origid,`matcher` FROM `resultdb`.`sm_matcher`;
END
```

```
delimiter
CREATE PROCEDURE init_table_sm_combi()
BEGIN
DROP TABLE IF EXISTS `sm_combi`;

CREATE TABLE `sm_combi` (
`id` smallint(6) NOT NULL,
`matcherids` varchar(30) NOT NULL,
PRIMARY KEY (`id`)
);

INSERT INTO `sm_combi` (`id`,`matcherids`)
VALUES (0, '');
END

delimiter
CREATE PROCEDURE init_table_sm_combi_agg(cid int)
BEGIN
#current aggid and counter
DECLARE agg int DEFAULT 0;
DECLARE ac int DEFAULT 0;
DECLARE done int DEFAULT FALSE;
DECLARE acur CURSOR FOR (SELECT `id` FROM `param_agg`);
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

DROP TABLE IF EXISTS `sm_combi_agg`;

CREATE TABLE `sm_combi_agg` (
`id` int(11) NOT NULL,
`combiid` smallint(6) DEFAULT NULL,
`sourceid` smallint(6) NOT NULL,
`aggid` smallint(6) NOT NULL,
`sqid` int(11) NOT NULL,
`tqid` int(11) NOT NULL,
`seid` int(11) NOT NULL,
`teid` int(11) NOT NULL,
`similarity` float(5.4) NOT NULL,
`count` smallint(6) NOT NULL,
PRIMARY KEY (`id`)
);

#now insert an entry for all diff q q e e pairs
#as init for dummy combi cid=0 and every agg
OPEN acur;

agg_loop: LOOP
FETCH acur INTO agg;

IF done THEN LEAVE agg_loop;
END IF;

INSERT INTO `sm_combi_agg`
(`id`,`combiid`,`sourceid`,`aggid`,`sqid`,`tqid`,
`seid`,`teid`,`similarity`,`count`)
```

```
(SELECT `id`+ac*@rcount,cid,`sourceid`,agg,`sqid`,`tqid`,
`seid`,`teid`,0,0
FROM `sm_result` #take first set of result tab
WHERE id <= @rcount);

SET ac = ac+1;
END LOOP;
CLOSE acur;
END

delimiter
CREATE PROCEDURE create_table_tmp()
BEGIN
DROP TABLE IF EXISTS `tmp`;

CREATE TABLE `tmp` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`sourceid` smallint(6) NOT NULL,
`sqid` int(11) NOT NULL,
`tqid` int(11) NOT NULL,
`seid` int(11) NOT NULL,
`teid` int(11)NOT NULL,
`similarity` float(5,4) NOT NULL,
PRIMARY KEY (`id`)
);
END
#m is origid,
#put result in tmp and select with given parameters
delimiter
CREATE PROCEDURE select_sm_result_to_tmp(m int, t float, d float, n int)
BEGIN

DELETE FROM `tmp` WHERE `id`>0;

INSERT INTO `tmp`
(`id`,`sourceid`,`sqid`,`tqid`,`seid`,`teid`,`similarity`)

SELECT CASE WHEN MOD(`id`,@rcount) = 0 THEN @rcount
ELSE MOD(`id`,@rcount) END,`sourceid`,`sqid`,`tqid`,`seid`,`teid`,
CASE WHEN `similarity`< t THEN 0 ELSE `similarity` END AS `similarity`

FROM `sm_result` WHERE `matcherid`=m;

#other selection strategies would be applied here...
END

delimiter
CREATE PROCEDURE sm_aggregate(combine bool)
BEGIN
#current matcher and count of matchers considered
DECLARE m int DEFAULT 0;
DECLARE mc int DEFAULT 0;
#counter to insert data in portions
DECLARE c int DEFAULT 0;
DECLARE cc int DEFAULT 0;
#params to select original match results
```

```
DECLARE o int DEFAULT 0;
DECLARE mco float DEFAULT 0;

CALL init_view_sm_matcher();
CALL init_table_sm_combi();
CALL init_table_sm_combi_agg(0);
CALL create_table_tmp();
SET mc = (SELECT COUNT(*) FROM `sm_matcher`);
SET m = 0;
SET @cid = 0;

#for each matcher
m_loop: LOOP

IF m = mc THEN LEAVE m_loop;
END IF;
#put all results (for all sources at once!) in tmp view
SET o = (SELECT `origid` FROM `sm_matcher` WHERE `id`=m);
#id of combi with only that matcher
SET mco = POW(2,m);

#selection
SET @t=0, @d=0, @n=0;
IF combine THEN SET @t = best_agg_t(m);
END IF;
CALL select_sm_result_to_tmp(m, @t, @d, @n);

#create new combis by extending all existing
INSERT INTO `sm_combi` (`id`,`matcherids`)

(SELECT @cid := @cid+1, CONCAT(`matcherids`,m,',') FROM `sm_combi`);

#add matcher result to create new combinations
############################## AVG
#for each combi already aggregated
SET c = 0;

c_loop: LOOP

IF c = mco THEN LEAVE c_loop;
END IF;

#offset of combis before + 1 (we have no offset for other aggs at the
    moment) = start id of current combi
SET @caggstart = c*@acount*@rcount+1;
SET @caggend = @caggstart+@rcount 1;

INSERT INTO `sm_combi_agg`
(`id`,`combiid`,`sourceid`,`aggid`,`sqid`,`tqid`,`seid`,`teid`,
`similarity`,`count`)

#@rcount*(a.combiid+mco) is offset to get into result range of the combi
SELECT tmp.id+@acount*@rcount*(a.combiid+mco),
#combiid+matcherid in dec is new combiid
a.combiid+mco,
a.`sourceid`,a.`aggid`,a.`sqid`,a.`tqid`,a.`seid`,a.`teid`,
```

```
a.`similarity`+tmp.`similarity`,
a.`count`+SIGN(tmp.`similarity`)
FROM
(SELECT *
FROM `sm_combi_agg`
# is same as:`combiid`=c AND `agg` LIKE 'AVG' (all sources at a time!)
WHERE id>=@caggstart AND id<=@caggend) a
#we cannot preselect similarity here, because we must insert all values
JOIN `tmp` tmp
ON tmp.`id`=a.`id` @caggstart+1;

SET c = c+1;
END LOOP;
SET m = m+1;
END LOOP;
#post process values
SET c = 1;
SET cc = (SELECT MAX(id) FROM `sm_combi`);
c_loop: LOOP

IF c > cc THEN LEAVE c_loop;
END IF;

SET @startid = c*@acount*@rcount+1;

UPDATE `sm_combi_agg`
SET
`similarity` = `similarity`/`count`
WHERE `count` > 0
AND `id` >= @startid AND id < @startid+@acount*@rcount;

SET c = c+1;
END LOOP;
END

#CALL sm_aggregate(true)

#combine.sql
#combines the element level results of schema matcher(s)
#(contained in table >specify!)
#to one overall value

#counts sm_Result size (=on element level)
SET @rcount = (SELECT SUM(sm_result_count(`id`)) FROM `param_src`);
#counts Source size (=on query level)
#get size over ids because is index column. same would be count(*)
SET @scount = (SELECT SUM(count*(count+1)/2)
FROM (SELECT MAX(id) MIN(id)+1 AS count FROM `query` GROUP BY `sourceid`)
    _);
#count of aggs currently considered in sm_combi_agg
SET @acount = (SELECT COUNT(*) FROM `param_agg`);
#nbr of thresholds used for computing combined schema sim
SET @tcount = 10;

delimiter
CREATE PROCEDURE create_table_sm_combi_result()
```

```
BEGIN
DROP TABLE IF EXISTS `sm_combi_result`;

CREATE TABLE `sm_combi_result` (
  `id` int(11) NOT NULL,
  `combiid` smallint(6) NOT NULL,
  `aggid` smallint(6) NOT NULL,
  `aggthreshold` float(4,3) NOT NULL,
  `sourceid` smallint(6) NOT NULL,
  `sqid` int(11) NOT NULL,
  `tqid` int(11) NOT NULL,
  `similarity` float(5,4) NOT NULL,
PRIMARY KEY (`id`)
);
END

delimiter
CREATE FUNCTION combined_similarity(startid int, endid int, t float,
sqec int, tqec int)
RETURNS float
BEGIN

SET @av=0,@sems=0,@tems=0;

SET @_= (SELECT 0 FROM
(SELECT @av :=AVG(`similarity`),@sems:=COUNT(DISTINCT `seid`),
                        @tems:=COUNT(DISTINCT `teid`)
FROM (SELECT * FROM resultdb.sm_result
        WHERE `id` >= startid and `id` <= endid) sqtqmapping
WHERE `similarity`>0 AND `similarity`>=t)_);

#is the case if no sim is > 0
IF @av IS NULL THEN SET @av = 0,@sems = 0,@tems = 0;
END IF;

RETURN @av * (@sems+@tems)/(sqec+tqec);
END

delimiter
CREATE PROCEDURE combine(cid int, agg int, ac int, src int, qc int,
sqorig int, sqec int, tqorig int,tqec int, startid int, endid int,
eval boolean, t float(4,3))
BEGIN

DECLARE v float(5,4) DEFAULT 0;
DECLARE id int(11) DEFAULT 0;

SET v = combined_similarity(startid,endid,t,sqec,tqec);
SET id = (ac*@scount) + qc;
INSERT INTO `sm_combi_result`
(`id`,`combiid`,`aggid`,`aggthreshold`,`sourceid`,`sqid`,`tqid`,`
    similarity`)
VALUES
(id, cid, agg, t, src, sqorig, tqorig, v);
```

```
END

#src param only to insert it in result table
delimiter
CREATE PROCEDURE select_combine(cid int, agg int, ac int, src int,
qc int, sqorig int, sqec int, tqorig int,tqec int, startid int, endid int
    ,
eval boolean)
BEGIN

DECLARE v float(5,4) DEFAULT 0;
DECLARE t float(4,3) DEFAULT 0;
DECLARE id int(11) DEFAULT 0;

IF @tcount>1 THEN SET t=0;
ELSE SET t=0.5;
END IF;

t_loop: LOOP

IF t > 0.9 THEN LEAVE t_loop;
END IF;

SET v = combined_similarity(startid,endid,t,sqec,tqec);

IF eval THEN SET id = (ac*@scount*@tcount) + t*10*@scount + qc;
ELSE SET id = ((cid 1)*@acount*@scount*@tcount)+(ac*@scount*@tcount) + t
    *10*@scount + qc;
END IF;

INSERT INTO `sm_combi_result`
(`id`,`combiid`,`aggid`,`aggthreshold`,`sourceid`,
`sqid`,`tqid`,`similarity`)
VALUES
(id, cid, agg, t, src, sqorig, tqorig, v);

IF @tcount = 1 THEN LEAVE t_loop;
END IF;
SET t = t+0.1;
END LOOP;
END

delimiter
CREATE PROCEDURE combine_sm_result(combine boolean, evaltab varchar(45),
    tevaltab varchar(45))
BEGIN
#loop params
DECLARE eval boolean DEFAULT (evaltab IS NOT NULL);

DECLARE cid smallint(6) DEFAULT 0;
DECLARE agg smallint(6) DEFAULT 0;
DECLARE src smallint(6) DEFAULT 0;
DECLARE sqorig int(11) DEFAULT 0;
DECLARE tqorig int(11) DEFAULT 0;
DECLARE sqecount int(11) DEFAULT 0;
DECLARE tqecount int(11) DEFAULT 0;
```

```
DECLARE tqstart int(11) DEFAULT 0;
DECLARE tqend int(11) DEFAULT 0;
DECLARE startid int(11) DEFAULT 0;
DECLARE endid int(11) DEFAULT 0;

DECLARE ac int DEFAULT 0;
DECLARE qc int DEFAULT 1;#counts ids!!

DECLARE done int DEFAULT FALSE;
DECLARE ccur CURSOR FOR (SELECT `id` FROM evaldb.`sm_combi`);
DECLARE acur CURSOR FOR (SELECT `id` FROM `param_agg`);
DECLARE qcur CURSOR FOR (
SELECT * ,
        @tqend+1 AS tstart, @tqend := @tqend+secount*tecount AS tend
FROM
        (SELECT s.`sourceid`,
                s.`origid` AS sorig,s.`ecount` AS secount,
                t.`origid` AS torig,t.`ecount` AS tecount
        FROM `query` s
        JOIN `query` t
        USING (`sourceid`)
        WHERE s.`origid` <= t.`origid`
        ORDER BY s.`sourceid` DESC,s.`origid`,t.`origid`)_
);
#we can declare only one event handler, so reset it after events
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

IF eval THEN
CALL create_table_eval(evaltab);
END IF;

CALL create_table_sm_combi_result();

OPEN ccur;
combi_loop: LOOP
FETCH ccur INTO cid;
IF done THEN LEAVE combi_loop;
END IF;

IF cid=0 OR (combine=FALSE AND MOD(LOG2(cid),1)>0) THEN ITERATE
    combi_loop;
END IF;

#start id of that combi's rows
#this start is also valid if combine was false for aggregation!
SET @cstart=@acount*@rcount*LOG2(cid);#is in tqstart incl+1;
#id offset multiplicator bec of agg
SET ac = 0;SELECT cid,@cstart;

OPEN acur;
agg_loop: LOOP
FETCH acur INTO agg;
IF done THEN
        SET done = FALSE;
        LEAVE agg_loop;
END IF;
```

```
SET @cstart=@cstart+@rcount*ac;
SET qc=1;
SET @tqend=0;

OPEN qcur;
q_loop: LOOP
FETCH qcur INTO src,sqorig,sqecount,tqorig,tqecount,tqstart,tqend;
IF done THEN
        SET done = FALSE;
        LEAVE q_loop;
END IF;

SET startid=@cstart+tqstart;
SET endid=@cstart+tqend;

IF tevaltab IS NULL THEN
#perform selection, combine, and insert in sm_combi_result
CALL select_combine(cid, agg,ac,src,qc,sqorig,sqecount,tqorig,tqecount,
    startid,endid,eval);
ELSE
SET @ath=best_agg_t(LOG2(cid));
CALL combine(cid, agg,ac,src,qc,sqorig,sqecount,tqorig,tqecount,startid,
    endid,eval,@ath);

END IF;

SET qc = qc+1;
END LOOP;
CLOSE qcur;
SET ac = ac+1;
END LOOP;
CLOSE acur;

IF eval THEN
        CALL eval_result('sm_combi_result',evaltab,true,false,tevaltab);
        #reset for next iteration
        DELETE FROM `sm_combi_result` WHERE `id`>0;
END IF;
END LOOP;
CLOSE ccur;
END

#CALL combine_sm_result(false,'sm_eval','sm_eval_config');

#eval.sql
#evaluates (and selects) a query matcher result

delimiter
CREATE PROCEDURE create_table_eval(tab VARCHAR(45))
BEGIN

CALL drop_table(tab);

SET @str = ' (
`id` int(11) NOT NULL AUTO_INCREMENT,
```

```
`matcherid` smallint(6) NOT NULL,
`sourceid` smallint(6) NOT NULL,
`aggid` smallint(6) NOT NULL,
`aggthreshold` float(4,3) NOT NULL,
`threshold` float(4,3) NOT NULL,
`pr` float(4,3) NOT NULL,
`re` float(4,3) NOT NULL,
`fm` float(4,3) NOT NULL,
`pr@3` float(4,3) NOT NULL,
`pr@5` float(4,3) NOT NULL,
`pr@10` float(4,3) NOT NULL,
`pr@15` float(4,3) NOT NULL,
`mae@3` float(4,3) NOT NULL,
`mae@5` float(4,3) NOT NULL,
`mae@10` float(4,3) NOT NULL,
`mae@15` float(4,3) NOT NULL,
`ndpm@3` float(4,3) NOT NULL,
`ndpm@5` float(4,3) NOT NULL,
`ndpm@10` float(4,3) NOT NULL,
`ndpm@15` float(4,3) NOT NULL,
PRIMARY KEY (`id`)
)';

SET @str = CONCAT('CREATE TABLE ', tab, @str);

PREPARE stmt FROM @str;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

END

delimiter
CREATE PROCEDURE create_view_gold(src int)
BEGIN
SET @s = src;#workaround since view def does not allow for variables

CREATE OR REPLACE VIEW `gold` AS
SELECT * FROM `goldstandard` WHERE `sourceid`=gv_src();

END

delimiter
CREATE FUNCTION gv_src()
RETURNS int
BEGIN

RETURN @s;
END

delimiter
CREATE FUNCTION eval_tmp_fast(t float)
RETURNS varchar(60)
NOT DETERMINISTIC
BEGIN
DECLARE pr1 float(5,4) DEFAULT 0;
DECLARE re1 float(5,4) DEFAULT 0;
```

```
DECLARE fm1 float(5,4) DEFAULT 0;
DECLARE pr3 float(5,4) DEFAULT 0;
DECLARE pr5 float(5,4) DEFAULT 0;
DECLARE pr10 float(5,4) DEFAULT 0;
DECLARE pr15 float(5,4) DEFAULT 0;

SET @t := t, @g=0.5, @pr:=0,@re:=0,@fm:=0;

SET @_ := (SELECT 0 FROM (
SELECT  @pr:=AVG(pr2),
@re:=AVG(re2),
@fm := AVG(2*(pr2*re2/(pr2+re2))),

@pr3:=AVG(`pr@3`),#we need no null check since denominator is k
@pr5:=AVG(`pr@5`),
@pr10:=AVG(`pr@10`),
@pr15:=AVG(`pr@15`)
FROM ( SELECT   *,
IF(pr IS NULL,0,pr) pr2,
IF(re IS NULL,1,re) re2#the 1 is correct since we do a right join with
    gold

FROM (SELECT `sqid`,
SUM(truepos)/SUM(pos) AS pr,
SUM(IF(rank<=3,truepos,0))/3 AS `pr@3`,
SUM(IF(rank<=5,truepos,0))/5 AS `pr@5`,
SUM(IF(rank<=10,truepos,0))/10 AS `pr@10`,
SUM(IF(rank<=15,truepos,0))/15 AS `pr@15`

#NO mae computable since we do not consider all in gold but only pos..
FROM (SELECT `sqid`,
        pos,
        IF(g.`similarity` IS NULL,0,pos) AS truepos,
        r.rank
FROM (SELECT *, IF(`similarity`>0 AND `similarity` >= @t,1,0) AS pos
                FROM `tmp`) r
LEFT JOIN (SELECT * FROM `gold` WHERE `similarity` >= @g) g
USING (`sourceid`,`sqid`,`tqid`))__
GROUP BY `sqid` ) prs
#we might not have recall vals for all
LEFT JOIN
(SELECT `sqid`,
SUM(CASE WHEN p.`similarity` IS NULL THEN 0 ELSE 1 END)/COUNT(*) AS re
FROM (SELECT * FROM `tmp` WHERE `similarity`>0 AND `similarity`>= @t) p
RIGHT JOIN (SELECT * FROM `gold` WHERE `similarity` >= @g) _g
USING (`sourceid`,`sqid`,`tqid`)
GROUP BY `sqid`) res
USING (`sqid`)
)__
)_);

IF @pr IS NULL THEN SET @pr = 0;
END IF;
IF @re IS NULL THEN SET @re = 0;
END IF;
IF @fm IS NULL THEN set @fm = 0;
```

```
END IF;

SET pr1=@pr, re1=@re, fm1=@fm, pr3=@pr3, pr5=@pr5, pr10=@pr10, pr15=@pr15;

RETURN CONCAT_WS(',', pr1, re1, fm1, pr3, pr5, pr10, pr15);
END

#assumes only one src set in tmp
#and corresponding goldstandard in gold
delimiter
CREATE FUNCTION eval_tmp_mae(k int, t float)
RETURNS float
NOT DETERMINISTIC
BEGIN

SET @k := k, @t := t;

SET @mae :=
(SELECT AVG(mae)
FROM (SELECT `sqid`, AVG(ABS(r.sim g.`similarity`)) AS mae
#final best k recs
FROM (SELECT *,
        #we calculate regarding selected sim...
        CASE WHEN `similarity`>= @t THEN `similarity` ELSE 0 END AS sim
        FROM `tmp`
        WHERE `rank` <= @k) r# recs
#joined with gold
JOIN `gold` g
USING (`sourceid`,`sqid`,`tqid`)
GROUP BY `sqid`) _
);

IF @mae IS NULL THEN SET @mae=0;
END IF;

RETURN @mae;
END

#assumes only one src set in tmp
#and corresponding goldstandard in gold
delimiter
CREATE FUNCTION eval_tmp_ndpm(k int, t float)
RETURNS float
NOT DETERMINISTIC
BEGIN
SET @k := k, @t := t;

#mean ndpm
/****/
SET @ndpm :=
(SELECT AVG(ndpm) FROM
(SELECT
SUM(
CASE WHEN g1.sqid is null
THEN IF(g2.sqid is null,1,2)     #l=r=0,l<r
ELSE CASE WHEN g2.sqid is null THEN 0 #l>r
```

```
ELSE IF(g1.similarity<g2.similarity, 2, IF(g1.similarity=g2.similarity
    ,1,0)) END END #AS nominator
)/(2*COUNT(*)) ndpm
FROM (
SELECT l.sqid, l.tqid AS lt, r.tqid AS rt
FROM (SELECT *,IF(`similarity`>=@t,`similarity`,0) AS sim FROM `tmp`
    WHERE rank<=@k) l
JOIN (SELECT *,IF(`similarity`>=@t,`similarity`,0) AS sim FROM `tmp`
    WHERE rank<=@k) r
USING (sqid)
WHERE l.tqid<>r.tqid AND l.sim>r.sim ) s
LEFT JOIN (SELECT * FROM `gold` WHERE rank<=@k AND similarity>0) g1
ON s.sqid=g1.sqid and lt=g1.tqid
LEFT JOIN (SELECT * FROM `gold` WHERE rank<=@k AND similarity>0) g2
ON s.sqid=g2.sqid AND rt=g2.tqid
GROUP BY s.sqid) _);

#should never be the case...
IF @ndpm IS NULL THEN SET @ndpm=0;
END IF;

RETURN @ndpm;
END

#evaltab the table where the eval result is inserted
delimiter
CREATE PROCEDURE eval_tmp(evaltab varchar(45), evalparams varchar(45), t
    float(4,3))
BEGIN

SET @str = CONCAT('INSERT INTO ', evaltab, ' VALUES (0');
SET @str = CONCAT_WS(',',@str,evalparams,t,eval_tmp_fast(t));

SET @str = CONCAT_WS(',',@str,eval_tmp_mae(3,t),eval_tmp_mae(5,t),
    eval_tmp_mae(10,t),eval_tmp_mae(15,t));
SET @str = CONCAT_WS(',',@str,eval_tmp_ndpm(3,t),eval_tmp_ndpm(5,t),
    eval_tmp_ndpm(10,t),eval_tmp_ndpm(15,t));

SET @str = CONCAT(@str, ');');

PREPARE stmt FROM @str;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

END

#evaltab the table where the eval result is inserted
delimiter
CREATE PROCEDURE select_eval_tmp(evaltab varchar(45), evalparams varchar
    (45))
BEGIN
DECLARE t float(4,3) DEFAULT 0;
DECLARE tmin float(4,3) DEFAULT 0.0;
DECLARE tmax float(4,3) DEFAULT 1.0;

#apply different selections
```

```
SET t = tmin;
t_loop: LOOP

IF t > tmax THEN LEAVE t_loop;
END IF;

CALL eval_tmp(evaltab, evalparams, t);

SET t = t+0.10;
END LOOP;
END

#evaltab the table to be created for eval data
#resulttab the table where the result comes from
delimiter
CREATE PROCEDURE eval_result(resulttab varchar(45),evaltab varchar(45),
issm boolean, makeevaltab boolean, tevaltab varchar(45))
BEGIN
DECLARE mid int DEFAULT 0;
DECLARE agg int DEFAULT 0;
DECLARE agt float(4,3) DEFAULT 0;
DECLARE src int DEFAULT 0;
DECLARE startid int DEFAULT 0;
DECLARE endid int DEFAULT 0;
#cols the col name of the matcher and those of one other param
#which should be used for discerning the result
#for grouping. insert \'\' as dummy string)
#if for cols empty string insert here only the id column;
DECLARE cols varchar(60) DEFAULT '';
DECLARE grouping varchar(60) DEFAULT '';

DECLARE done int DEFAULT FALSE;
DECLARE rcur CURSOR FOR (SELECT * FROM tmp_cursor_view);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

#a view to get the ids to iterate over the results
#create the view for the cursor dynamically
IF issm THEN
SET cols=''combiid`,`aggid`,`aggthreshold``,
        grouping=''combiid`,`aggid`,`aggthreshold``;
ELSE SET cols=''matcherid`,1,0',grouping=''matcherid``;
END IF;

SET @str = 'CREATE OR REPLACE VIEW tmp_cursor_view as SELECT ';
SET @str = CONCAT(@str, cols,',`sourceid`,MIN(id),MAX(id) FROM ',
    resulttab,' GROUP BY ',grouping,',`sourceid`');

PREPARE stmt FROM @str;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;

#to record the result of one matcher temporarily
CALL create_table_qqsim('tmp');

#to record the eval results/***/
```

```
IF makeevaltab THEN
CALL create_table_eval(evaltab);
END IF;

SET @cursrc = 1;
OPEN rcur;
result_loop: LOOP
FETCH rcur INTO mid,agg,agt,src,startid,endid;
IF done THEN LEAVE result_loop;
END IF;

IF @cursrc<>src THEN
CALL create_view_gold(src);
SET @cursrc=src;
END IF;

#reset table tmp
DELETE FROM `tmp` WHERE `id`>0;
#get the result of the matcher/combi
CALL init_table_qqsim('tmp',resulttab,startid,endid,issm);

IF tevaltab IS NULL THEN
CALL select_eval_tmp(evaltab, CONCAT_WS(',',mid,src,agg,agt));
ELSE
Set @tx=0;
CALL best_t(mid,agt,tevaltab, @tx);
CALL eval_tmp(evaltab, CONCAT_WS(',',mid,src,agg,agt), @tx);
END IF;
END LOOP;
CLOSE rcur;

DROP VIEW tmp_cursor_view;
END

#CALL eval_result('resultdb.result','eval3',false,true,'')
```

B. List of Figures

C. Bibliography

[1] Apache lucene. http://lucene.apache.org/.

[2] Apache tomcat. http://tomcat.apache.org/.

[3] Aqua data studio 8.0. 5.9.5 auto completion. http://www.aquafold.com/.

[4] Business objects. http://www.sap.com/solutions/sapbusinessobjects/large/business-intelligence/index.epx.

[5] Cognos software. http://cognos.com/.

[6] Databasespy sql editor. efficiently edit complicated sql queries. http://www.altova.com/databasespy.html.

[7] Eclipse modeling framework project (emf). http://www.eclipse.org/modeling/emf/.

[8] Gene ontology. http://www.geneontology.org/.

[9] Glossary of data mining terms. olap. http://webdocs.cs.ualberta.ca/ zaiane/courses/cmput690/glossary.html.

[10] Iso/iec 9075-part 1-4,9-11,13, and 14. information technology – database languages – sql.

[11] Junit. http://www.junit.org/.

[12] Lucene 2.9.0 api. - class similarity.

[13] Mysql. http://www.mysql.com/.

[14] Sloan digital sky survey. http://www.sdss.org/dr8/.

[15] Tpc benchmarks. http://www.tpc.org/information/benchmarks.asp.

[16] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. Sql querie recommendations. *Proc. VLDB Endow.*, 3:1597–1600, September 2010.

[17] J. Akbarnejad, M. Eirinaki, S. Koshy, D. On, and N. Polyzotis. Sql querie recommendations: a query fragment-based approach. In *4th InternationalWorkshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB '10)*, 2010.

[18] M. Barile. Taxicab metric. from mathworld – a wolfram web resource, created by e. w. weisstein. http://mathworld.wolfram.com/taxicabmetric.html.

[19] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. volume 4, pages 695–701, 2011.

[20] P. Raghavan C. D. Manning and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8:215–260, 1997.

[22] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. S. V. Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[23] W. W. Chu and G. Zhang. Associative query answering via query feature similarity. In *Proc. IIS*. IEEE Press, 1997.

[24] S. Cohen, W. Nutt, and Y. Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54, April 2007.

[25] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching a comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.

[26] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 330–341. Morgan Kaufmann, 1996.

[27] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 259–270, New York, NY, USA, 1998. ACM.

[28] H.-H. Do. *Schema Matching and Mapping-based Data Integration*. PhD thesis, University of Leipzig, 8 2005.

[29] H.-H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002.

[30] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the 30th international conference on Very large data bases*, volume 30 of *VLDB '04*, pages 372–383. VLDB Endowment, 2004.

[31] M. Dumas, L. Garcia-banuelos, and R. Dijkman. Similarity search of business process models. *IEEE Data Eng. Bull.*, 32(3):23–28, 2009.

[32] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, 2007.

[33] S. Finkelstein. Common expression analysis in database applications. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, SIGMOD '82, pages 235–245, New York, NY, USA, 1982. ACM.

[34] T. Gaasterland. Cooperative answering through controlled query relaxation. *IEEE Expert: Intelligent Systems and Their Applications*, 12:48–59, September 1997.

[35] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa. Plan selection based on query clustering. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 179–190. VLDB Endowment, 2002.

[36] A. Giacometti, P. Marcel, and E. Negre. A framework for recommending olap queries. In *Proceedings of the ACM 11th international workshop on Data warehousing and OLAP*, 2008.

[37] A. Giacometti, P. Marcel, and E. Negre. Recommending multidimensional queries. In *DaWaK*, pages 453–466, 2009.

[38] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for olap discovery-driven analysis. *IJDWM*, 7(2):1–25, 2011.

[39] A. Giordani and A. Moschitti. Semantic mapping between natural language questions and sql queries via syntactic pairing. In *Natural Language Processing and Information Systems*, volume 5723 of *Lecture Notes in Computer Science*, pages 207–221. Springer Berlin / Heidelberg, 2010.

[40] R. D. Gopal and R. Ramesh. The query clustering problem: A set partitioning approach. *IEEE Trans. on Knowl. and Data Eng.*, 7:885–899, 1995.

[41] M. Thiele Gunnar Schröder G. Schröder and W. Lehner. Setting goals and choosing metrics for recommender setting goals and choosing metrics for recommender system evaluations. In *the Proceedings of the Workshop on User-Centric Evaluation of Recommender Systems and Their Interface (UCERSTI), 5th ACM Recommender Systems conference (RecSys 2010)*, 2011.

[42] S. Guo, W. Sun, and M. A. Weiss. Solving satisfiability and implication problems in database systems. *ACM Trans. Database Syst.*, 21:270–293, June 1996.

[43] A. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, December 2001.

[44] R. W. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J*, 1950.

[45] D. Harman, E. Fox, R. Baeza-Yates, and W. Lee. *Inverted files. In Information Retrieval: Algorithms and Data Structures*, chapter 3, pages 28–43. Prentice-Hall, 1992.

[46] F. Hausdorff. *Grundzüge der Mengenlehre*, pages 32–33. Veit & Comp., Leipzig, 1914.

[47] Y. E. Ioannidis and S. D. Viglas. Conversational querying. *Information Systems*, 31(1):33 – 56, 2006.

[48] H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh. Preference-based recommendations for olap analysis. In *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '09, pages 467–478, Berlin, Heidelberg, 2009. Springer-Verlag.

[49] M. Drosou K. Stefanidis and E. Pitoura. "you may also like" results in relational databases. In *VLDB*, 2009.

[50] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *CIDR*, 2009.

[51] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: context-aware autocompletion for sql. volume 4, pages 22–33. VLDB Endowment, October 2010.

[52] N. Khoussainova, Y. Kwon, W.-T. Liao, M. Balazinska, W. Gatterbauer, and D. Suciu. Session-based browsing for more effective query reuse. In *Proceedings of the 23rd international conference on Scientific and statistical database management*, SSDBM'11, pages 583–585, Berlin, Heidelberg, 2011. Springer-Verlag.

[53] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.

[54] P. Koutris and E. Soroush. A machine learning approach to mapping natural language to sql. Technical report, University of Washington (CSE department), 2010.

[55] R. A. Olshen L. Breiman, J. H. Friedman and C. J. Stone. *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[56] G. Li, J. Fan, H. Wu, J. Wang, and J. Feng. Dbease: Making databases user-friendly and easily accessible. In *CIDR*, pages 45–56, 2011.

[57] G. Li, J. Feng, X. Zhou, and J. Wang. Providing built-in keyword search capabilities in rdbms. *The VLDB Journal*, 20(1):1–19, February 2011.

[58] A. Senart M. Seguran and D. Trastour. remix : A semantic mashup application. *META4eSociety 2012*, 2012.

[59] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[60] P. Marcel. Query personalisation and recommendation in data warehouse: anoverview. 2011.

[61] P. Marcel and E. Negre. A survey of query recommendation techniques for data warehouse exploration. In *7èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2011), Clermont-Ferrand*, volume B-7 of *RNTI*, pages 119–134, Paris, Juin 2011. Hermann.

[62] K. Mershad and H. Artail. Codisc: Collaborative and distributed semantic caching for maximizing cache effectiveness in wireless networks. *J. Parallel Distrib. Comput.*, 71:495–511, March 2011.

[63] T. Millstein, A. Halevy, and M. Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, 66(1):20 – 39, 2003.

[64] C. Monz and M. de RijkeSpring. Inverted index construction. introduction to information retrieval, 2002.

[65] N. Nihalani, S. Silakari, and M. Motwani. Natural language interface for database-a brief review. *International Journal of Computer Science Issues*, 8:600–608, 2011.

[66] N. Ohsugi, A. Monden, and K. Matsumoto. A recommendation system for software function discovery. In *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, 2002.

[67] J. Park and S. Lee. Keyword search in relational databases. *Knowledge and Information Systems*, February 2010.

[68] E. Peukert, J. Eberius, and E. Rahm. Amc - a framework for modelling and comparing matching systems as matching processes. In *Proc. Int. Conf. on Data Engineering*, 2011.

[69] E. Peukert, S. Maßmann, and K. König. Comparing similarity combination methods for schema matching. In *GI Jahrestagung (1)*, 2010.

[70] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[71] Q. Ren, M. H. Dunham, and V. Kumar. Semantic caching and query processing. *IEEE Trans. on Knowl. and Data Eng.*, 15:192–210, January 2003.

[72] D. J. Rosenkrantz and H. B. Hunt III. Processing conjunctive predicates and queries. In *Proceedings of the sixth international conference on Very Large Data Bases - Volume 6*, pages 64–72. VLDB Endowment, 1980.

[73] N. Roussopoulos, C. M. Chen, S. Kelley, A. Delis, and Y. Papakonstantinou. The adms project: Views "r" us, 1995.

[74] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 249–260, New York, NY, USA, 2000. ACM.

[75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[76] C. Sammut and G. I. Webb, editors. *Encyclopedia of Machine Learning*. Springer, 2010.

[77] C. Sapia. On modeling and predicting query behavior in olap systems. In *DMDW*, page 2, 1999.

[78] C. Sapia. Promise: Predicting query behavior to enable predictive caching strategies for olap systems. In *DaWaK*, pages 224–233, 2000.

[79] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13:23–52, March 1988.

[80] G. Shani and A. Gunawardana. Evaluating recommendation systems. In *Recommender Systems Handbook*, pages 257–297. 2011.

[81] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 2001.

[82] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 2nd edition, 2008.

[83] A. Vancea and B. Stiller. Coopsc: A cooperative database caching architecture. In *WETICE*, pages 223–228, 2010.

[84] S. Wartik. *Chapter 12: Boolean Operations. Information Retrieval Data Structures & Algorithms*. Prentice-Hall, 1992.

[85] E. W. Weisstein. Distance. from mathworld – a wolfram web resource. http://mathworld.wolfram.com/distance.html.

[86] P.-A. Larson Y. Silva and J. Zhou. Exploiting common subexpressions for cloud query processing. In *ICDE*, 2012.

[87] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 964–975, Washington, DC, USA, 2009. IEEE Computer Society.

[88] Y. Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *J. Am. Soc. Inf. Sci.*, 1995.

[89] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 105–116, New York, NY, USA, 2000. ACM.

[90] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Kluwer, 2006.

[91] J. Zhou, P.-A. Larson, J.-C. Freytag, and W. Lehner. Efficient exploitation of similar subexpressions for query processing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 533–544, New York, NY, USA, 2007. ACM.

Confirmation

I confirm that I independently prepared the thesis and that I only used the references and auxiliary means indicated in the thesis.

Dresden, August 31, 2012