Data-Thirsty Business Analysts need SODA - Search Over DAta Warehouse

Lukas Blunschi
ETH Zurich, Switzerland
Lukas.Blunschi@inf.ethz.ch

Claudio Jossen
Credit Suisse AG, Switzerland
Claudio.R.Jossen@credit-suisse.com

Donald Kossmann
ETH Zurich, Switzerland
Donald.Kossmann@inf.ethz.ch

Magdalini Mori
Credit Suisse AG, Switzerland
Magdalini.Mori@credit-suisse.com

Kurt Stockinger
Credit Suisse AG, Switzerland
Kurt.Stockinger@credit-suisse.com

ABSTRACT

Querying large data warehouses is very hard for non-tech savvy business users. Deep technical knowledge of both SQL as well as the schema of the database is required in order to build correct queries and to come up with new business insights. In this paper we introduce a novel system called SODA (Search Over DAta Warehouse) that bridges the gap between the business world and the IT world by enabling extended keyword search in a data warehouse. SODA uses metadata information, DBpedia entries as well as base data to generate SQL to allow intuitive exploration of the data. The process of query classification, query graph generation and SQL generation is visualized to provide the analysts with information on how the query results are produced. Experiments with real data of a global financial institution comprising around 300 tables showed promising results.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*information filtering, query formulation*

General Terms

Algorithms

Keywords

Data Warehouse, Metadata, Query Generation

1. INTRODUCTION

The goal of SODA is to enable business analysts to run ad-hoc queries on complex data warehouses such as those deployed in financial institutions. The ultimate goal of this project is to take a natural language query of a business analyst as input and automatically generate the correct SQL query that can be executed in the data warehouse.

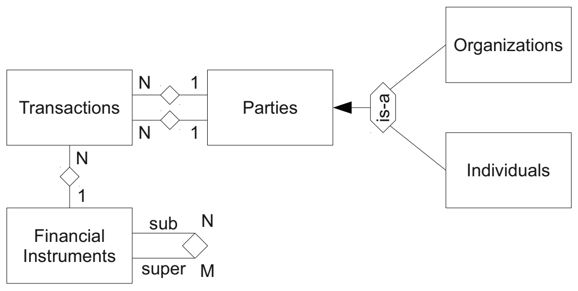


Figure 1: Sample World Schema

At the heart of SODA is a graph that represents the metadata of the data warehouse (i.e., the schema) as well as the business objects and concepts used by business analysts. Literally, this graph bridges the two worlds by providing paths from the business objects to the tables and attributes of the data warehouse. Furthermore, this graph integrates data from external data sources such as DBpedia. Finally, the graph integrates data from the data warehouse itself because queries might involve such instance data.

One important feature of SODA is that it is incremental. We do not expect to have a perfect meta-model from the beginning. Instead, the graph will be modified continually, thereby capturing all the experience gained from answering ad-hoc queries. Furthermore, we expect to evolve the algorithms that navigate the graph continually as more and more query patterns become apparent from using the system.

In summary, this paper reports on the following contributions:

- A comprehensive framework to model metadata and integrate other data sources for searching business entities in data warehouses.
- The design and implementation of a system that takes a business analyst query and a metadata graph as input and generates a SQL query as output.
- A visual user interface that explains the lineage of query results and allows a user to select the most appropriate result.

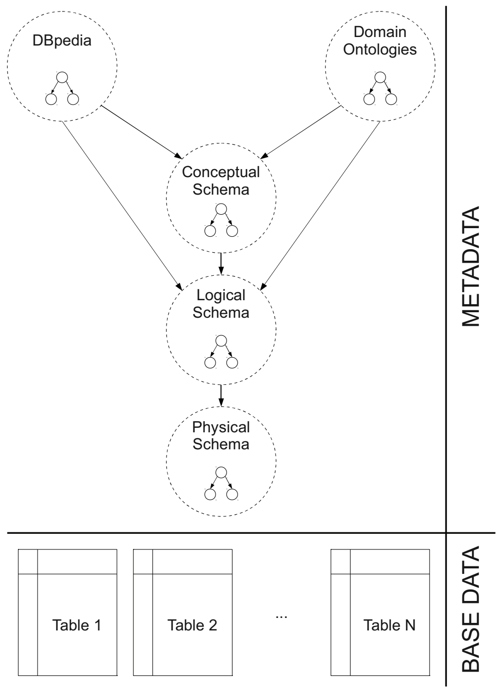


Figure 2: Metadata Graph and Relational Data

2. SAMPLE WORLD

For privacy reasons, we use a synthetic dataset which captures some of the main features of a real enterprise data warehouse. Figure 1 shows the schema of our sample world. In particular, we model information about customers (referred to as *Parties*) and the transactions these customers make; buying and selling on the stock market. Parties can be individuals for private banking or corporate customers for investment banking (i.e., *Organizations*); both kinds of parties are modeled separately because they are supported by different sets of analysts. The technical term for *products* which can be bought or sold on the stock market is *Financial Instrument*. Financial instruments can be shares of a company (e.g., IBM shares). Financial instruments, however, can also be structured; that is, a financial instrument could relate to a funds that manages a portfolio of shares or even a hedge fund that manages a portfolio of certificates of other funds and hedge funds. It is in part this recursive nature of financial instruments that makes it difficult for business analysts to extract the right information from a data warehouse, if no pre-canned reports exist that serve the needs of the analyst.

3. SYSTEM OVERVIEW

In this section we provide a brief overview of our system and explain the algorithms by means of the following query:
`customers Zürich financial instruments`

3.1 Input Data

Our data warehouse consists of base data stored in a relational database as well as metadata stored in a graph structure (such as RDF). The metadata consists of the database schema extended with DBpedia entries and domain ontologies (see Figure 2).

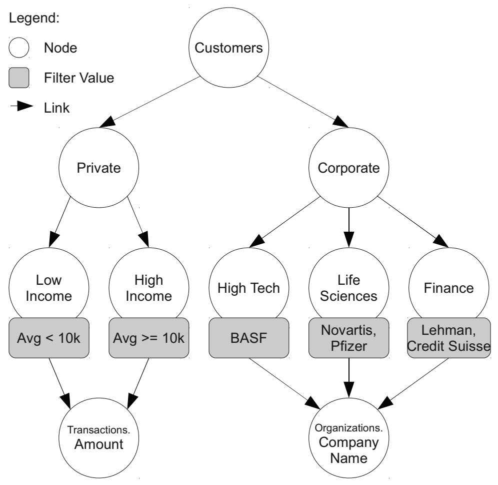


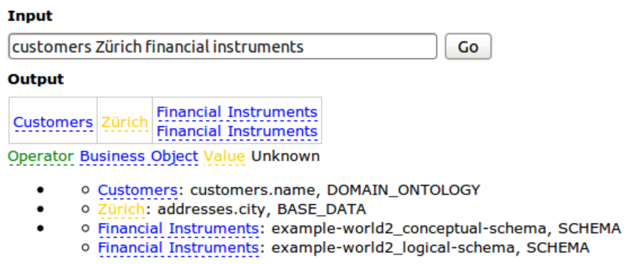
Figure 3: Customers Domain Ontology

Integrated Schema A data warehouse combines and aggregates data from many heterogeneous data sources. To handle the differences in the data sources, an integrated schema is built. This schema can become very large and complex on the technical level. Therefore, there are different levels of the schema, namely conceptual, logical and physical. The conceptual schema (business layer) serves for communication with business and contains the main entities to be modeled such as parties, securities and accounts. The logical schema extends the conceptual one by showing inheritance, splitting entities (for instance, parties are split into individuals and organizations), etc. The physical schema contains information about database indexes or table partitioning. Typically all these schemas are designed with one modeling tool with the goal to generate the physical tables.
DBpedia The metadata graph also contains an extract from DBpedia. Note that we only extract those DBpedia entries that have direct connections to the terms stored in our database schema. For instance, for the term "Parties" shown in our example world, the following entries might be extracted from DBpedia: customer, client, political organization etc. Now, when a user searches for customers then parties would be an answer.
Domain Ontologies Besides the schema and DBpedia, our metadata also consists of several domain ontologies. The domain ontologies are built specifically for a given data warehouse and are used to classify data for a specific domain.

In our example world we are using one ontology that classifies financial instruments and one that classifies customers (see Figure 3). In this example, customers are divided into private and corporate customers. Private customers are further classified as "low income" and "high income" customers based on their total transaction amount. On the other side, corporate customers are categorized as "high tech", "life sciences" or "finance".
Base Data The base data is stored in a relational database.

3.2 Query Classification

The very first step in order to answer our sample query is to interpret the keyword query and apply it onto the metadata graph, before actually executing queries on the data warehouse. In fact, the syntax of the natural language query


Input

customers Zürich financial instruments

Output

Customers	Zürich	Financial Instruments Financial Instruments

Operator Business Object Value Unknown

- Customers: customers.name, DOMAIN_ONTOLOGY
- Zürich: addresses.city, BASE_DATA
- Financial Instruments: example-world2_conceptual-schema, SCHEMA
  - Financial Instruments: example-world2_logical-schema, SCHEMA

Figure 4: Query Classification

needs to be mapped to a formal syntax to be used for SQL generation (see Figure 4). We call this *query classification*. Query classification applies the following object types to a natural language query:

Business Objects Each single word as well as sequences of words are looked up in the metadata graph and if there is a match, then the term or term combination is classified as a business object. The underlying assumption is that in this case the business object is describing an aspect that can be used as an entry point for a graph traversal to an actual schema node.

Operators Some single words and short concatenations of words in a predefined list are used for a check for operators.

Values If a keyword appears in the base data, we consider it a value. In order to identify the table column that contains the respective value, we use an inverted index.

Unknown Every word or word combination that cannot be classified as one of the three mentioned object types is considered as filler due to the use of natural language and is ignored for the further processing of the query.

Figure 4 shows a screen shot of the classification of our sample query. The term "customers" is classified as a business object which is defined in the customers domain ontology. "Zürich" was found in the base data and "financial instruments" is a business object found in the conceptual and logical schema.

3.3 Algorithm Flowchart

Figure 5 shows the main algorithmic steps starting from query classification (lookup) and ending with SQL generation. Each of these steps is described in more detail in the following sub-sections.

Lookup During the lookup phase, we identify the words in the given query. To do that, we lookup each word in the classification index which is an inverted index on all the words of the input data (metadata graph and relational data). A lookup of a single word provides us with all the locations where this word is found. The output of the lookup phase provides us with entry points into the metadata graph where we can continue applying our algorithm. In Figure 4 "customers" and "Zürich" are both found only once, whereas "financial instruments" is found twice. These two possible combinations result in two solutions.

Rank and Top N In this phase, we compute a first score for all the possible solutions from the lookup phase. There are many possible heuristics which can be used here. One such heuristic is to make use of the location where a word was found. E.g. a word which was found in the domain ontology gets a higher score than a word which was found in DBpedia. The output of the rank and top N phase adds a

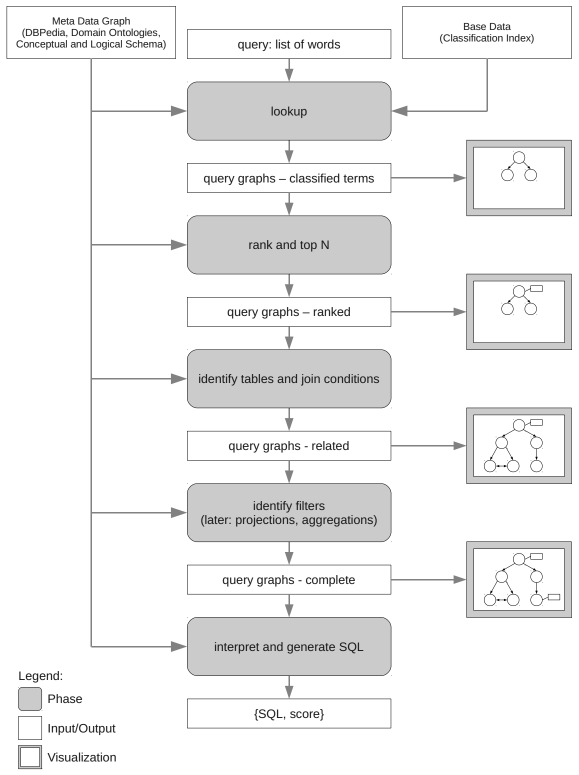


Figure 5: Algorithm Flowchart

score to the solutions found in the lookup phase, orders the results according to that score and continues with the top N results.

Tables and Joins The purpose of this phase is to identify all the tables which are used in each solution and to discover the relationships between these tables. Starting at every entry point which we discovered in the lookup phase, we follow all the edges in the metadata graph until we either have no more edges to follow or discover a table. To find the connections between these tables, we look for graph patterns which represent relationships between tables, e.g. primary-foreign key links and inheritance patterns. The output of the tables and joins phase adds tables and their relationships to the solutions.

In our example, we have two solutions and both of them contain four separate SQL queries. One of the queries has the following tables:
`addresses, fi_transactions, financial_instruments, individuals, parties, transactions`

Filters In the filter phase we look for filter conditions which should be applied to each solution. A filter condition consists of a column and a value such as "Zürich". To define these filter conditions, we look at all the paths from the entry points to the columns. The output of the filters phase adds filter conditions to the previously discovered columns.

In our example, the filter conditions are used to connect "Zürich" to the city column within the addresses table.

SQL Generation During this phase, we generate SQL for all the solutions. To create the SQL statement, we make use of all the tables, relationships and filter conditions which were discovered in the previous phases. The foreign keys are used to combine the tables. If the tables are partitioned into distinct subsets (via inheritance patterns), then a separate SQL statement is created for each subset, in the query above, for example, we get a SQL statement for individuals and one for organization:

```
SELECT *
FROM addresses, individuals, fi_transactions,
     financial_instruments, parties, transactions
WHERE
  addresses.city='Zürich' AND
  (transactions.fromParty=parties.id OR
   transactions.toParty=parties.id) AND
  fi_transactions.id=transactions.id AND
  fi_transactions.fi=financial_instruments.id AND
  individuals.id=parties.id AND
  individuals.livingAt=addresses.id

SELECT *
FROM fi_transactions, financial_instruments,
     organizations, parties, transactions
WHERE
  (organizations.companyName='BASF' OR ... OR
   organizations.companyName='Credit Suisse') AND
  (transactions.fromParty=parties.id OR
   transactions.toParty=parties.id) AND
  organizations.id=parties.id AND
  fi_transactions.id=transactions.id AND
  fi_transactions.fi=financial_instruments.id
```

The output of the SQL phase is a list of SQL statements for each solution together with a score.

4. MORE DEMO INSIGHTS

In addition to generating SQL based on keywords, our system also provides various result visualization features that help the user understand and interpret the results. For example, a user can look at the lineage for each term in the classified query or he (or she) may browse along the edges of the metadata graph.

Experiments with real data of a global financial institution comprising around 300 tables showed promising results. For privacy reasons this demo is based on the synthetic sample world shown in Section 2.

Our demo will provide answers to the following questions:

- How is a query classified and broken into meaningful entities that can be used for further processing?
- How are the query results ranked?
- How can tables and join conditions be identified and visualized?
- How are filter conditions evaluated and how are they use for generating useful SQL?
- How are range queries and aggregations performed?

At `http://lukasblunschi.ch/soda` you can find a demo version of SODA using the sample world of Section 2.

5. RELATED WORK

Due to the popularity of search engines and the ease of data exploration, adding search capabilities to relational databases has become very popular [1, 3, 4, 5, 6, 7, 8, 2].

Whereas these works mainly make use of the database schema to discover joins, our metadata graph not only consists of one or several combined schemas, but can integrate domain-specific extensions (domain ontologies) as well as extensions which are not related to any specific domain (DBpedia).

Furthermore, one of the main differentiators of our work is that we consider inheritance within the database schema - a common practice in large and complex data warehouse schemas. The problems is important for automatic SQL generation. For instance, if we want to join the table transaction with the table parties where a party can be either an individual or an organization, we need to bear in mind the exclusive inheritance between individual and organization. A typical wrong join path is as follows: transaction AND individual AND organization. Taking into account the inheritance yields the following two join paths: transactions AND organization; transactions AND individual. None of the related work appears to consider inheritance.

Finally, existing work is validated against fairly simple data models with some 10 tables, while our system is based on a real-world data set with a complex enterprise data warehouse model of a global financial institution comprising hundreds of tables.

Hence, our approach is an important step to automatically generate correct SQL for complex, real-world data warehouse schemas.

6. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. *ICDE*, 2002.

[2] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis. Keyword Search over Relational Databases: A Metadata Approach. *SIGMOD*, 2011.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. *ICDE*, 2002.

[4] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. *SIGMOD*, 2007.

[5] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, 2002.

[6] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: Top-k Keyword Query in Relational Databases. *SIGMOD*, 2007.

[7] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano. Efficient Keyword Search Across Heterogeneous Relational Databases. *ICDE*, 2007.

[8] Q. Su and J. Widom. Indexing Relational Database Content Offline for Efficient Keyword-Based Search. *IDEAS*, 2005.