ProQua: A System for Evaluating Logic-Based Scoring Functions on Uncertain Relational Data

Sebastian Lehrack, Sascha Saretz and Christian Winkel
Brandenburg University of Technology Cottbus
Institute of Computer Science
Postfach 10 13 44, D-03013 Cottbus, Germany
{slehrack, ssaretz, cwinkel}@informatik.tu-cottbus.de

ABSTRACT

ProQua is an innovative probabilistic database system which enables the application of logic-based and weighted similarity conditions on uncertain relation data. In this demonstration paper we describe the interrelations among the main concepts, present an archaeological example scenario and sketch the software architecture of ProQua.

1. MOTIVATION

Our novel probabilistic database system *ProQua*[1] combines information retrieval techniques with database technologies. In the last Claremont report[2] [**?**] leading database researchers have proposed such a combination as an important and promising research field. Classical database systems evaluate a given query against a data tuple either to the truth value *true* or to the truth value *false*. This *strict* evaluation approach is not able to meet user expectations about vague and uncertain conditions adequately [**?**]. Consequently, there is a need for incorporating the concepts of impreciseness and proximity into a traditional database query language.

A promising approach for handling vagueness is exploiting *similarity conditions* as '*price about 100*' or '*location is close to Berlin*' within a *logic-based* query language. This type of query language combines similarity predicates by logical operators ($\wedge, \vee$ and $\neg$) in order to formulate *complex* conditions. Data tuples fulfill these complex conditions to a certain degree which is given by a *score value* taken from the interval $[0, 1]$.

Logic-based scoring functions represent the theoretical concept behind complex similarity conditions [**?**]. In general, they can be applied on a *certain* or an *uncertain* databases.

[1]ProQua stands for "probabilistic and quantum logic-based database system".

[2]The Database Research Self-Assessment Meeting takes place every five years.

EDBT/ICDT '13 March 18 - 22 2013, Genoa, Italy

Probabilistic databases have been established as a new type of database systems which can manage and query huge data sets of uncertain data [**?**].

In [**?**, **?**] we have published a probabilistic data and query model which enables the application of logic-based scoring functions on probabilistic databases. In addition, we implemented our techniques in ProQua as an extension of the Oracle database system [**?**]. In contrast to other state-of-the-art probabilistic database systems ProQua offers a generic similarity operator *within* its query language QSQL2 [**?**, **?**]. Furthermore, QSQL2 facilitates the weighting of subqueries and subconditions exploiting two different weighting approaches [**?**, **?**, **?**].

In the next section we discuss a schema of the key concepts of ProQua. Afterwards, we present the example scenario of our online demonstration in Sec. (3). Finally, we give an overview of the architectural components of ProQua in Sec. (4).

2. BASIC CONCEPTS

In this section we briefly discuss the main idea behind ProQua. For this purpose, we describe the interrelations among its core concepts, see Fig. (1).

In the first step we map the different syntactical components of a given QSQL2 query to (i) a logic-based scoring function and (ii) an algebra query on probabilistic data. Each query type is based on its own semantic model. On the one hand, a logic-based scoring function is interpreted by a probabilistic view of a vector space retrieval model [**?**, **?**]. On the other hand, we exploit the well-known possible-worlds-semantics for processing an algebra query on probabilistic data. Respecting these semantic models the outcome of a logic-based scoring function is determined by a query plan computing the result of a normalised similarity domain calculus query [**?**]. In contrast, the evaluation of an algebra query is performed by a query plan with intensional and extensional plan operators [**?**]. Those plans rely on a novel representation system for probabilistic databases called *U*-databases*. By employing the ranking semantics of *expected scores* [**?**] and a top-k filter [**?**] we finally generate an overall query plan which produces the desired query result.

3. EXAMPLE SCENARIO

In order to demonstrate the key features of ProQua and its query language QSQL2 we provide an online demo:

http://dbis.informatik.tu-cottbus.de/ProQua/.

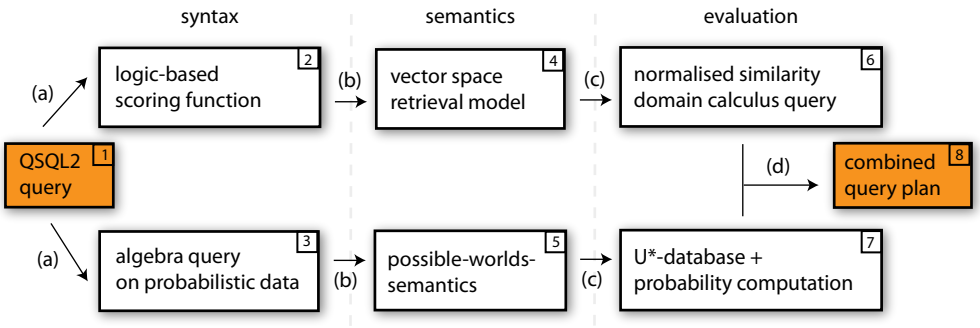

Figure 1: Main concepts of ProQua

The example scenario of this online demo is motivated by the redesign of the *CISAR* project[3]. It is a web-based Geo Information System for archaeology and building history [**?**]. Technologies of ProQua will be widely used in its successor system *OpenInfRA* [**?**].

Our simplified example scenario relies on the deterministic table *Artefacts* (*Arte*) and the two probabilistic tables *Artefacts classified by Experts* (*ArteExp*) and *Artefacts classified by Material* (*ArteMat*), see Fig. (3). In table *Arte* we store information about several artefacts which were found during an archaeological excavation. Thereby, the *sondage number* (attribute *sond*) of an artefact encodes its geographical area of finding.

Furthermore, various experts gave a valuation about the originating culture (attribute *culture*) for each artefact, see table *ArteExp*. Those estimations are quantified by a confidence value (attribute *conf*) embodying the probability that the considered artefact is created by the specified culture. Besides these subjective expert reports we also involve more objective methods. These *archaeometrical methods* (e.g. XRF and ICS-MS[4]) utilize material analyses. In combination with the artefact finding site and the age, the material composition can be useful to elucidate the originating culture. The corresponding confidence values are also given in the attribute *conf*.

Using those data tables our online demo contains several examples queries. All example queries are presented according to a classification of all supported query types. This classification is inferred from two independent criteria, namely (i) the expressiveness of the supported conditions and (ii) the nature of the underlying relational data basis. Thus, we indicate

- the capability of incorporating the concepts of impreciseness and vagueness in terms of similarity conditions, i.e. classical Boolean conditions (`BC`) vs. similarity conditions (`SC`), and
- the capability of expressing several possible database states derived from tuple and attribute uncertainty, i.e. classical certain data bases (`CD`) vs. uncertain data bases (`UD`).

By combining these criteria orthogonally we build the four query classes as described subsequently. They all are exemplified by a characteristic query taken from our example scenario[5].

Arte

aid	type	sond	age
art1	vase fragment	3	300
art2	spear head	10	500
art3	vase fragment	4	300

ArteExp

expert	rep	aid	culture	conf
Peter	B	art1	roman	0.3
Peter	B	art1	greek	0.4
Cathy	C	art1	roman	0.4
John	A	art2	egyptian	0.6

ArteMat

method	year	aid	culture	conf
XRF	1997	art1	roman	0.3
XRF	1997	art1	greek	0.3
ICS-MS	2008	art2	punic	0.8
XRF	2010	art2	egyptian	0.5

Figure 3: Data tables of the example scenario: *Arte*, *ArteExp* and *ArteMat*

(I.) Boolean conditions on certain data: The class `BConCD` includes queries formed by Boolean conditions (i.e. the result is given by either the truth value true or false) on deterministic relational data. These queries are supported by traditional relational query languages as SQL. Referring to our scenario a typical query of `BConCD` is exemplified by *"Determine all artefacts which have a sondage number between 7 and 12 or an age between 250 and 350 years"*. This kind of query produces a homogeneous result set of artefacts matching the defined condition.

(II.) Similarity conditions on certain data: The class `SConCD` represents queries which enable vagueness and impreciseness by similarity conditions. The result of such a query is defined by a *list of tuples* ordered by score values from the interval $[0, 1]$. These score values express the degree of query fulfilment. For instance, a `SConCD`-query is given by *"Determine all artefacts such that their sondage numbers should be around 10* or *their ages should be around 300 years"*.

(III.) Boolean conditions on uncertain data: The

[3]http://www.dainst.org/en/project/cisar/

[4]The abbreviations XRF and ICS-MS stand for the *x-ray fluorescence* and the *inductively coupled plasma mass spectrometry method*, respectively.

[5]In our online demo each query class is characterised by a weighted *and* an unweighted version of an example query. For the sake of brevity, we give in this paper just an unweighted query per query class.

```
select aid, type, culture
from    ( select aid, culture
          from ArteExp
        union[ 0.9, 0.4 ]
          select aid, culture
          from ArteMat
        ) origin
    inner join
        ( select *
          from Arte
          where ( sond ~ 10 or[ 0.3,
                  0.8 ] age ~ 300 )
        ) prop
    on ( origin.aid = prop.aid )
```

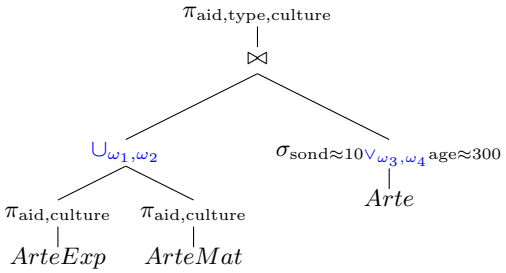

Figure 2: Example scenario query as QSQL2 query (left) and as abstract version (right)

queries of the class BConUD are typical for probabilistic databases. As an example query we give *"Determine all artefacts which are probably created by Roman people"*. Once again the resulting tuples are ranked. But in this case probabilities are used for the final ranking instead of score values.

(IV.) Similarity conditions on uncertain data: If we combine both criteria, then we achieve a query class with an expanded expressiveness. A SConUD-query is formulated as *"Determine all artefacts with their possible cultural origins such that their finding sondages should be near sondage* 10 *or their ages should be around* 300*"*. The ranking is now determined by a given ranking semantics, e.g. expected scores.

In order to give an example query in QSQL2 we present the last SConUD-query as weighted version in Fig. (2). In addition to similarity conditions, a user individualises this query by weighting the importance of various parts of the query on two different levels. Concretely, the user quantifies the significances between following query parts:

(i) the culture estimated by an expert vs. the culture classified by the material analysis (see keyword *union[0.9, 0.4]* and operator $\cup_{\omega_1,\omega_2}$ in Fig. (2)) and
(ii) the *sondage* vs. the *age* (see keyword *or[0.3, 0.8]* and operator $\vee_{\omega_3,\omega_4}$ in Fig. (2)).

For this query we assume that the considered user determines the weighting variables to $\omega_1 = 0.9, \omega_2 = 0.4, \omega_3 = 0.3$ and $\omega_4 = 0.8$. That means, the user prioritises (i) the expert valuation over the material analysis and (ii) the age over the sondage, because we define that a high weight from the interval $[0, 1]$ corresponds to a high priority.

4. PROQUA ARCHITECTURE

In this section we briefly sketch the architecture of the ProQua system (see Fig. 4). In detail, we present the main components of ProQua by describing the primary control and data flow.

The foundation of our implementation is a transformation sequence of different *operator plans*. The operators within these plans are developed as specialised versions of the well-known relational algebra operators. By adapting algebra operators we facilitate a direct and efficient integration into an existing object-relational database system like Oracle 11g [**?**]. The current ProQua implementation consists of (i) a client library for Java applications and (ii) a cartridge which extends the Oracle DBMS by scoring and probability computation capabilities.

The components (1) to (6) of Fig. (4) form the client library *QSQL2forJava*. This library replaces the database driver which is usually invoked by a Java application. In contrast, the components (7) to (8) are directly integrated within the Oracle DBMS. In the following we briefly describe the main components of ProQua as depicted in Fig. (4).

(1) QSQL2 syntax engine: The two main tasks of the *QSQL2 syntax engine* are the syntax checking and the construction of an abstract syntax tree.

(2) sf plan generator: The *sf plan generator* constructs a logic-based scoring function by processing the incoming syntax tree. Subsequently, the *sf plan generator* carries out a syntactical normalisation as pre-step for constructing a *sf plan*. It can be shown that each sf plan can compute score values in polynomial time.

(3) pr plan generator: The *pr plan generator* also takes the abstract syntax tree and extracts an algebra query which can be evaluated by a *pr plan*. The underlying *extensional* and *intensional* plan operators calculate the probabilities for all resulting tuples. In general, the computation of those probabilities is in the complexity class $\#\mathcal{P}$[6].

(4) sf/pr plan combiner: The *sf/pr plan combiner* merges the sf plan and the pr plan to an overall query plan. Thereby, the applied combining strategy intensively depends on the underlying ranking semantics. In the current implementation ProQua exploits the expected score semantics.

(5) sf/pr plan optimiser: Even though the complexity class of the merged query plan is determined by its input plans, the optimiser can reach significant improvements of the overall processing time by using further standard optimisation strategies on the plan operator level.

(6) plan2Oracle transformer: The last transformation step is the compilation of the optimised query plan into (several) Oracle query statements by a special transformer component tailored for the supported Oracle DBMS. Referring to the modular architecture of ProQua we are able to support other systems and platforms by developing further transformer modules for such systems and platforms.

(7) ProQua query engine: The *ProQua query engine* is responsible for the computation of all score values and probabilities. Thus, the various plan operators can call scoring functions and probability computation algorithms provided by the ProQua query engine to determine the final result.

(8) Oracle query engine: Besides the computation of score values and probabilities delivered by (7) the *Oracle query engine* creates the relational data part of the computed query result.

[6]Problems of the complexity class $\#\mathcal{P}$ ask for the number of accepting paths of a non-deterministic Turing machine which is running in polynomial time (i.e. $\mathcal{NP} \subset \#\mathcal{P}$).

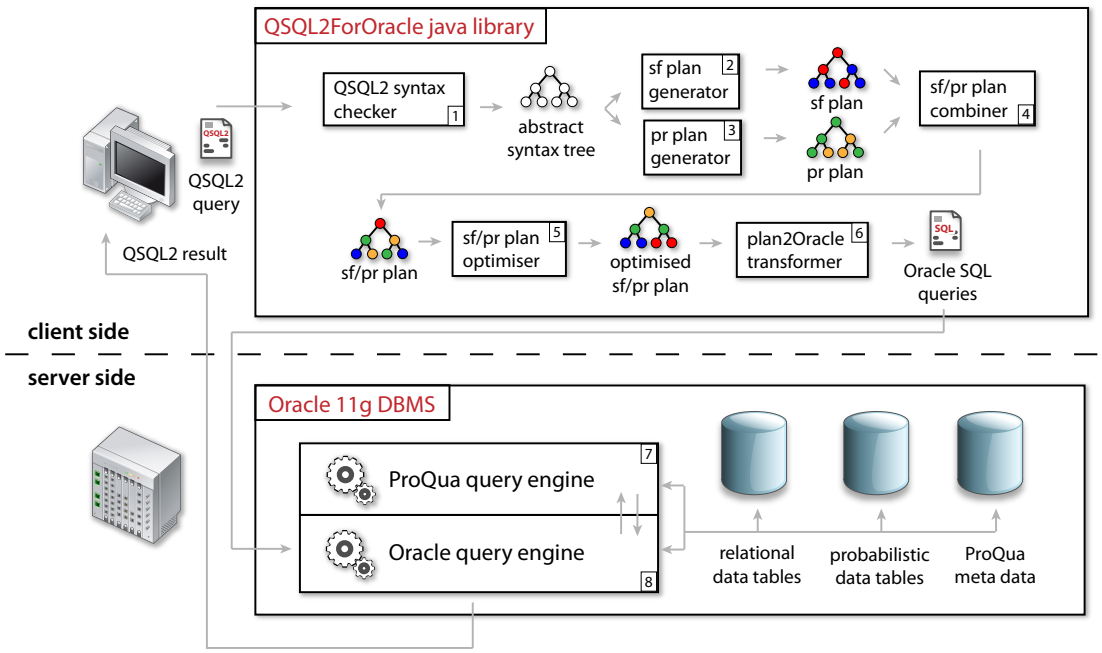

Figure 4: Architecture of the ProQua database system

Acknowledgment: The development of ProQua was supported by the German Research Foundation (DFG) grants SCHM 1208/11–1 and SCHM 1208/11–2.

5. REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[2] Frank Henze, Heike Lehmann, and Wolfgang Langer. CISAR - A Modular Database System as a Basis for Analysis and Documentation of Spatial Information. In *CAA*, pages 228–233, 2007.

[3] Ihab F. Ilyas and Mohamed A. Soliman. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on DM. Morgan & Claypool, 2011.

[4] Sebastian Lehrack. Applying weighted queries on probabilistic databases. In *CIKM*, pages 2209–2213, 2012.

[5] Sebastian Lehrack and Sascha Saretz. A Top-k Filter for Logic-Based Similarity Conditions on Probabilistic Databases. In *ADBIS*, pages 268–281, 2012.

[6] Sebastian Lehrack and Sascha Saretz. Evaluating Logic-Based Scoring Functions on Uncertain Relational Data. *JIDM*, 3(3):348–363, 2012.

[7] Sebastian Lehrack, Sascha Saretz, and Ingo Schmitt. QSQL2: Query Language Support for Logic-Based Similarity Conditions on Probabilistic Databases. In *RCIS*, pages 1–12, 2012.

[8] Sebastian Lehrack and Ingo Schmitt. QSQL: Incorporating Logic-Based Retrieval Conditions into SQL. In Hiroyuki Kitagawa, Yoshiharu Ishikawa, Qing Li, and Chiemi Watanabe, editors, *DASFAA (1)*, volume 5981 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2010.

[9] Sebastian Lehrack and Ingo Schmitt. A Probabilistic Interpretation for a Geometric Similarity Measure. In *ECSQARU*, pages 749–760, 2011.

[10] Sebastian Lehrack and Ingo Schmitt. A Unifying Probability Measure for Logic-Based Similarity Conditions on Uncertain Relational Data. In *Proceedings of the International Workshop on New Trends in Similarity Search (EDBT workshop)*, pages 14–19, New York, NY, USA, 2011. ACM.

[11] Tony Morales. *Oracle Database Reference, 11g Release 1 (11.1)*. Oracle Corp., Publishers, 2008.

[12] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[13] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein et al. The Claremont report on database research. *SIGMOD Rec.*, 37:9–19, September 2008.

[14] Felix Schäfer and Alexander Schulze. OpenInfRA – Storing and retrieving information in a heterogenous documentation system. In *CAA*, 2012.

[15] Ingo Schmitt. Weighting in CQQL. BTU Cottbus, Computer Science Reports 04/07, 2007.

[16] Ingo Schmitt. QQL: A DB&IR Query Language. *VLDB J.*, 17(1):39–56, 2008.

[17] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[18] David Zellhöfer and Ingo Schmitt. A preference-based approach for interactive weight learning: learning weights within a logic-based query language. *Distributed and Parallel Databases*, 27(1):31–51, 2010.