SIGMOD Officers, Committees, and Awardees

Chair

Yannis Ioannidis
University of Athens
Department of Informatics
Panepistimioupolis, Informatics Bldg
157 84 Ilissia, Athens
HELLAS
+30 210 727 5224
<yannis AT di.uoa.gr>

Vice-Chair

Christian S. Jensen
Department of Computer Science
Aarhus University
Åbogade 34
DK-8200 Århus N
DENMARK
+45 99 40 89 00
<csj AT cs.aau.dk >

Secretary/Treasurer

Alexandros Labrinidis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260-9161
PA 15260-9161
USA
+1 412 624 8843
<labrinid AT cs.pitt.edu>

SIGMOD Executive Committee:

Sihem Amer-Yahia, Curtis Dyreson, Christian S. Jensen, Yannis Ioannidis, Alexandros Labrinidis, Maurizio Lenzerini, Ioana Manolescu, Lisa Singh, Raghu Ramakrishnan, and Jeffrey Xu Yu.

Advisory Board:

Raghu Ramakrishnan (Chair), Yahoo! Research, <First8CharsOfLastName AT yahoo-inc.com>,
Amr El Abbadi, Serge Abiteboul, Rakesh Agrawal, Anastasia Ailamaki, Ricardo Baeza-Yates, Phil Bernstein, Elisa Bertino, Mike Carey, Surajit Chaudhuri, Christos Faloutsos, Alon Halevy, Joe Hellerstein, Masaru Kitsuregawa, Donald Kossmann, Renée Miller, C. Mohan, Beng-Chin Ooi, Meral Ozsoyoglu, Sunita Sarawagi, Min Wang, and Gerhard Weikum.

Information Director, SIGMOD DiSC and SIGMOD Anthology Editor:

Curtis Dyreson, Washington State University, <cdyreson AT eecs.wsu.edu>

Associate Information Directors:

Denilson Barbosa, Ugur Cetintemel, Manfred Jeusfeld, Georgia Koutrika, Alexandros Labrinidis, Michael Ley, Wim Martens, Rachel Pottinger, Altigran Soares da Silva, and Jun Yang.

SIGMOD Record Editor:

Ioana Manolescu, INRIA Saclay, <ioana.manolescu AT inria.fr>

SIGMOD Record Associate Editors:

Magdalena Balazinska, Denilson Barbosa, Pablo Barceló, Vanessa Braganholo, Chee Yong Chan, Ugur Çetintemel, Brian Cooper, Cesar Galindo-Legaria, Glenn Paulley and Marianne Winslett.

SIGMOD Conference Coordinator:

Sihem Amer-Yahia, Qatar Computing Research Institute, <sihemameryahia AT acm.org>

PODS Executive: Maurizio Lenzerini (Chair), University of Roma 1, <lenzerini AT dis.uniroma1.it>,
Phokion G. Kolaitis, Jan Paradaens, Thomas Schwentick, Jianwen Su and Dirk Van Gucht.

Sister Society Liaisons:

Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment).

Awards Committee:

Laura Haas (Chair), IBM Almaden Research Center, <laura AT almaden.ibm.com>, Rakesh Agrawal, Peter Buneman, and Masaru Kitsuregawa.

Jim Gray Doctoral Dissertation Award Committee:

Johannes Gehrke (Co-chair), Cornell Univ.; Beng Chin Ooi (Co-chair), National Univ. of Singapore, Alfons Kemper, Hank Korth, Alberto Laender, Boon Thau Loo, Timos Sellis, and Kyu-Young Whang.

[Last updated : October 10th, 2011]

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Until 2003, this award was known as the "SIGMOD Innovations Award." In 2004, SIGMOD, with the unanimous approval of ACM Council, decided to rename the award to honor Dr. E. F. (Ted) Codd (1923 - 2003) who invented the relational data model and was responsible for the significant development of the database field as a scientific discipline. Recipients of the award are the following:

Michael Stonebraker (1992)
David DeWitt (1995)
Serge Abiteboul (1998)
Rudolf Bayer (2001)
Ronald Fagin (2004)
Jennifer Widom (2007)
Umeshwar Dayal (2010)
Jim Gray (1993)
C. Mohan (1996)
Hector Garcia-Molina (1999)
Patricia Selinger (2002)
Michael Carey (2005)
Moshe Y. Vardi (2008)
Surajit Chaudhuri (2011)
Philip Bernstein (1994)
David Maier (1997)
Rakesh Agrawal (2000)
Don Chamberlin (2003)
Jeffrey D. Ullman (2006)
Masaru Kitsuregawa (2009)

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)
Jeffrey Ullman (1996)
Raghu Ramakrishnan (1999)
Daniel Rosenkrantz (2001)
Surajit Chaudhuri (2004)
Hans-Jörg Schek (2007)
David Lomet (2010)
Gio Wiederhold (1995)
Avi Silberschatz (1997)
Michael Carey (2000)
Richard Snodgrass (2002)
Hongjun Lu (2005)
Klaus R. Dittrich (2008)
Gerhard Weikum (2011)
Yahiko Kambayashi (1995)
Won Kim (1998)
Laura Haas (2000)
Michael Ley (2003)
Tamer Özsu (2006)
Beng Chin Ooi (2009)

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* This award, which was previously known as the SIGMOD Doctoral Dissertation Award, was renamed in 2008 with the unanimous approval of ACM Council in honor of Dr. Jim Gray. Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau, University of Washington. *Runners-up*: Marcelo Arenas, University of Toronto; Yanlei Diao, University of California at Berkeley.
- **2007** *Winner*: Boon Thau Loo, University of California at Berkeley. *Honorable Mentions*: Xifeng Yan, University of Indiana at Urbana Champaign; Martin Theobald, Saarland University
- **2008** *Winner*: Ariel Fuxman, University of Toronto. *Honorable Mentions*: Cong Yu, University of Michigan; Nilesh Dalvi, University of Washington.
- **2009** *Winner*: Daniel Abadi, MIT. *Honorable Mentions*: Bee-Chung Chen, University of Wisconsin at Madison; Ashwin Machanavajjhala, Cornell University.
- **2010** *Winner:* Christopher Ré, University of Washington. *Honorable Mentions*: Soumyadeb Mitra, University of Illinois, Urbana-Champaign; Fabian Suchanek, Max-Planck Institute for Informatics.
- **2011** *Winner*: Stratos Idreos, Centrum Wiskunde & Informatica. *Honorable Mentions*: Todd Green, University of Pennsylvania; Karl Schnaitter, University of California in Santa Cruz.

A complete listing of all SIGMOD Awards is available at: **http://www.sigmod.org/awards/**

[Last updated : October 10th, 2011]

Editor's Notes

Welcome to the March 2012 issue of the ACM SIGMOD Record! This is a rich issue with many interesting contributions.

This issue opens with an article by Chang on the optimization of XML twig pattern queries featuring full-text search predicates. The optimization of XML twig pattern queries has received significant attention in the past; at the same time, full-text search is very useful in text-rich XML databases, and a significant extension of XQuery specifically deals with full-text search. The article explores a space of optimization techniques exploiting together the structural and text constraints of the query, proposing a cost model and optimization heuristics.

In the Database Principles column, the paper by Peter Wood surveys query languages for graph databases, a topic on which attention is renewed due to the popularity of important applications such as Semantic Web graphs of RDF data, and social network graph analysis. Query languages are analyzed from the viewpoint of features (expressive power) and then under the angle of their associated algorithmic complexity.

Two surveys appear in this issue. The first one, by Dustdar, Pichler, Savenkov and Truong, attempts to systematize an area at the confluence of data management and Web services, namely data services. The authors argue for a new approach in modeling, describing and integrating distributed and heterogeneous data sources, by means of quality-aware data services, and discuss issues they raise, such as metadata management, data service publication, and the selection of the "best" services to be used for a given integration task. The survey by Mami and Bellahsene considers the problem of recommending views to materialize in order to improve the performance of a given query workload; the authors analyze and classify many algorithms from the existing literature from the angle of the general view selection framework they use (data cube lattice, ANR-OR view graph etc.) as well as from the cost constraint perspective.

The issue includes two Distinguished Profiles in Databases. The first one features David Lomet, whose outstanding career has been distinguished among others by two ACM SIGMOD Best Paper awards, as well as being an ACM Fellow and an IEEE Fellow. The column is remarkable in many respects, of which I'll just select two. First, beyond the many successful projects, successes, we also learn of decisions and technical views that David came to regret later on, which I find reassuring for the rest of us mortals! Second, the interview smoothly takes you quite advanced points of detail concerning transactions, isolation, latches, all the way up to flash disks and cloud computing. Our second distinguished columnist is Catriel Beeri, also an ACM Fellow. His interview blends the story of his research career between Jerusalem, Toronto, and Princeton, advising Moshe Vardi and sharing a room with Phil Bernstein, with beautiful stories of ideas, such as data dependencies and the chase. Read it to find out the full implications and connections between the data dependency theory, data integration, data-exchange, view-based query evaluation and more – and also the possibly lesser-known music lover side of Catriel!

The Research Centers column features a presentation of the Institute for the Management of Information Systems, within the "Athena" Research center, headed by Timos Sellis. The article is dense with descriptions of the IMIS' research areas: Data lifecycles, Geoinformatics, Web of Data, Biological Data Management, Privacy and more!

The Industry Perspective column features an article on the new SQL:2011 standard, issued in December 2011. Although these days a lot of attention is given to systems claiming to be not (or not only!) SQL, one should still not ignore the elephant in the living room – namely, the still-dominant industry standard

language. The article by Zemke outlines the extensions brought by the new standard, such as: enhancement of collection types, pipelined data manipulations and more.

Finally, in the Open Forum column, Grossniklaus and Maier outline the curriculum in cloud-based data management at Portland State University, in Oregon, USA. The authors detail their choices of courses, and student projects, and recommended readings. In this very fast-changing landscape of technology, this presentation is timely as many University departments are pondering the organization of similar courses.

This year's SIGMOD conference will be held early, on May 22-24 in Scottsdale, Arizona, as usual jointly with PODS and several interesting workshops. I'll be there and looking forward to any feedback you may have on the Record's overall organization and content! We will also be looking for new editors, contact me at the conference if you are interested.

Your contributions to the Record are welcome via the RECESS submission site (http://db.cs.pitt.edu/recess). Prior to submitting, be sure to peruse the Editorial Policy on the SIGMOD Record's Web site (http://www.sigmod.org/publications/sigmod-record/sigmod-record-editorial-policy).

Ioana Manolescu

March 2012

Past SIGMOD Record Editors:

Harrison R. Morse (1969)
Daniel O'Connell (1971 – 1973)
Randall Rustin (1975)
Thomas J. Cook (1981 – 1983)
Jon D. Clark (1984 – 1985)
Margaret H. Dunham (1986 – 1988)
Arie Segev (1989 – 1995)
Jennifer Widom (1995 – 1996)
Michael Franklin (1996 – 2000)
Ling Liu (2000 – 2004)
Mario Nascimento (2005 – 2007)
Alexandros Labrinidis (2007 – 2009)

Optimizing XML Twig Queries with Full-Text Predicates

Ya-Hui Chang
Department of Computer Science and Engineering
National Taiwan Ocean University, Keelung, 202, Taiwan, R.O.C.
E-mail: yahui@ntou.edu.tw

ABSTRACT

Efficient query processing has been a critical issue for XML repositories. In this paper, we consider the XML query which can be represented as a query tree with twig patterns, and also consists of full-text constraints. Previously, the *structure-first* approach and the *keyword-first* approach have been proposed to process such kind of queries. The main focus of this paper is constructing an integrated system to support these two approaches and find the best execution plan. To achieve this goal, we first analyze the components of these two approaches and design a set of operators. We then derive the corresponding cost model and rewriting rules to perform cost-based optimization. We also propose several heuristic rules by observing the behaviors of the two approaches. Via an extensive experimental study, we demonstrate that our cost-based system and heuristic system are both effective.

1. INTRODUCTION

As the XML (eXtensible Markup Language) technology emerged as the de facto standard for information sharing and data exchange on the Web, XML data management and query processing have attracted a lot of attention from the academic and business communities.

In general, the nested structure of an XML document is captured by a tree model, so XML queries, *e.g.*, XPath or XQuery, are normally specified based on path expressions to navigate the complex structure of XML data. The path expressions involved in a query might constitute a tree structure, and such query is usually named as a *twig query*. There have been many research efforts on the relevant processing techniques [2, 3, 4, 9]. They usually first identify the elements which meet the tag or path requirements, and apply specially-designed encoding schemes, algorithms, or data structures, *e.g.*, indices or stacks, to expedite the combining process.

In contrast, researchers also advocate keyword-based search against XML documents, since it provides a more friendly user environment. Specifically, users do not need to explicitly state the structural constraint, but the system will ensure that the returning data satisfy the structure of the queried XML document [1, 7, 8, 11]. This type of researches usually apply the techniques seen in the information retrieval (IR) field, *e.g.*, inverted lists or ranking schemes.

In view of the need to combine the above two querying facilities, XQuery and XPath Full-Text (XQFT) 1.0, has been proposed and became a W3C standard [10]. Consider the following sample query, which is posed against the XML document in Figure 1:

Q1
for $p in /catalog/item
where $p/remark **contains text** (“database” ftand “design” **ordered**) ftand (“database” ftand “design” **distance** at least 2 words) and $p//name **contains text** (“Peter” ftand “Rob” **ordered**)
return $p

In this query, the path expressions “*/catalog/item*”, “*/catalog/item/remark*” and “*/catalog/item//name*” form a twig pattern, which represent the *structural constraint* on the retrieved data. On the other hand, the operator *“contains”* is used to enforce the content of the elements *remark* and *name* to satisfy the given *full-text constraints*. Particularly, the content of the element “/catalog/item/remark” is required to have an ordering between the keywords “database” and “design”, with the distance at least “2” between them.

We have previously proposed two ways to process XML queries with these constraints [3]. The first one is called the *structure-first* approach. It mainly follows the research direction originally designed for processing twig queries, but is extended to be able to process full-text constraints as well. In contrast, the second one is called the *keyword-first* approach, since it is mainly based on the techniques for processing keyword-based queries. This approach will first process the full-text constraints, and then make its answer also satisfy the structural constraint. Although these two approaches have been shown both feasible, they were built as two separate systems where users are hard to determine which one to use.

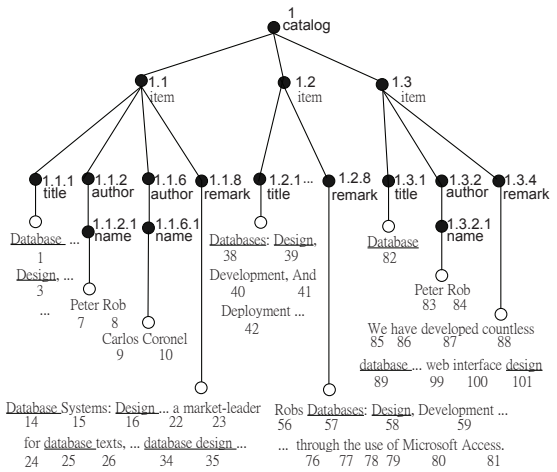


Figure 1: The Sample XML Tree

In this paper, we intend to construct an integrated system to support the *structure-first* (SF) approach and the *keyword-first* (KF) approach in a uniform way. We first analyze the components of these two approaches and design a set of operators to represent how they work. We then define the cost model to represent the cost of each operator and propose a set of rewriting rules, so that we can choose the plan with the least cost from all possible execution plans. In addition to supporting cost-based optimization, we also propose several heuristic rules by observing the behaviors of the two approaches. Finally, we have conducted a series of empirical studies. The experimental results show the effectiveness of the cost-based optimization system and the proposed heuristic rules.

The remaining of this paper is organized as follows. We first introduce the underlying data model and the two approaches in Section 2. We then describe how we achieve cost-based optimization in Section 3. We have performed a series of experiments to demonstrate the effectiveness of our integrated system. The experimental results are analyzed in Section 4. Finally, conclusion and future works are discussed in Section 5.

2. PRELIMINARIES

In this section, we first discuss the underlying data representation in our system. We then explain the SF approach and the KF approach. The XML tree in Figure 1 and the sample query $Q1$ will be used as the running example throughout this paper.

2.1 Data Modeling

The XML document is represented as a rooted labeled tree as usual. To quickly determine the structural relationship between two elements, each element node is associated with the extended Dewey encoding [9]. There are mainly two reasons for applying this encoding scheme. First, this encoding scheme has the main properties of the original Dewey encoding, so we can easily obtain the LCA (lowest common ancestor) of two elements by computing their common prefixes. This is required by the keyword-first approach. For example, the LCA of elements 1.1.1 and 1.1.2 is 1.1. Second, a unique property of the extended Dewey encoding is that it can be easily transformed to a labeled path. For example, encoding 1.1.1 can be transformed to the path "/catalog/item/title". This supports the efficient processing of twig joins.

For processing full-text constraints, each keyword in the context is also given a *global* position, which is assigned across elements. For example, the first "Database" under the leftmost *title* element is encoded as 1, while the first "Database" under the leftmost *remark* element is encoded as 14.

2.2 The Structure-First Approach

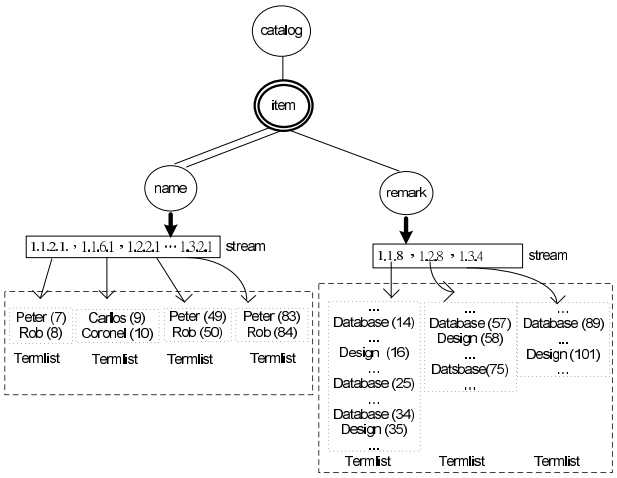


Figure 2: The Query Tree with Streams and Termlists

In the SF approach, an input query is first constructed as a query tree to clearly show its structural constraint. The query tree is built based on all the path expressions specified in the query, where the component elements are illustrated by nodes and the location steps are denoted by edges. The node with a double circle indicates the answer node. After the query tree is built, the SF approach first identifies the elements which match the tag constraint for each leaf node of the query tree and represents them in a sorted order, which are called the *stream* of the associated node. For each element in the stream, the SF approach then represents the component keywords along with their positions in a sorted order, which are named as the *Termlists*. Figure 2 illustrates the result of processing *Q1* against the sample XML tree at this stage.

The SF approach then examines each element in the stream, and uses a variant of the merge-sort algorithm, to see if the associated Termlist represents the queried keywords in the required *ordered* or *distance* sequence. For example, in the stream of the node *name*, the elements 1.1.2.1, 1.2.2.1, and 1.3.2.1 all satisfy the *ordered* constraint for the keywords *Peter* and *Rob*. After identifying all the elements which satisfy the full-text constraint from each individual stream, the SF approach applies a twig join algorithm, called TJFast [9], to find the correct answers which satisfy the whole twig constraint.

2.3 The Keyword-First Approach

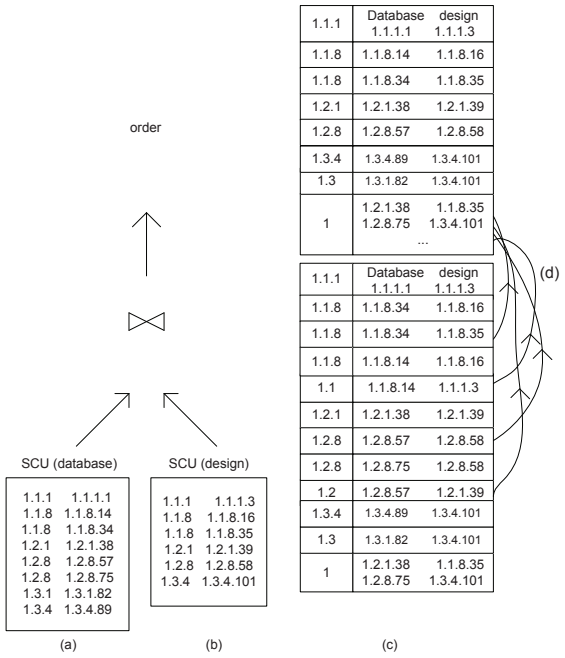


Figure 3: An Example of the KF Approach

In contrast to Termlists, the underlying data structure of the KF approach is the SCU (Smallest Containing Unit) [1]. An SCU consists of a list of items, where each item represents the element and the position which matches the pattern (keyword or full-text constraint). For example, in Figure 3(a)-(b), we can see the SCU tables for the keywords "database" and "design", where the position is represented by attaching the global position to the extended Dewey encoding of its parent element. Another difference from the SF approach is that the items in an SCU table are sorted based on the postorder of nodes, instead of preorder.

The KF approach starts by processing the full-text constraints. For each full-text constraint which consists of the keywords K1 and K2, it will first create the corresponding SCU tables (Figure 3(a)-(b)). It then calculates the LCAs which represent both keywords K1 and K2, as seen in Figure 3(c). To identify those elements which satisfy the full-text constraint, the algorithm first determines if an LCA satisfies the given constraint. If it does not, the matched position will be sent to its nearest ancestor to check next. For example, as shown in Figure 3(d), element 1.1 does not satisfy the first ordered constraint in $Q1$, because the matched position for "database" (1.1.8.14) is bigger than the matched position for "design" (1.1.1.3). Therefore, these information will be propagated to its ancestor (element 1) for further checking. Note that in this approach, it is comparably easy to check the full-text predicates *ordered* and *distance*, since we can directly retrieve the matched positions for the keywords from the SCU, and a simple arithmetic calculation will suffice. After obtaining those elements which satisfy the full-text constraints, the KF approach will represent them as streams of the corresponding leaf node in the query tree, and find answers by invoking the TJFast algorithm.

3. COST-BASED OPTIMIZATION

In this section, we discuss how we combine the SF and KF approaches into an integrated system, and how to perform cost-based optimization.

3.1 Operators

Table 1: Operators

operator	cost model
T(t)	$c_1 * \|T(t)\|$
K(k)	$c_2 * \|K(k)\|$
$C_{op}^{t1,t2}$(.)	$c_3 * \|I\|$; I: input Termlist
	$c_4 * \|I\|$; I: input SCU
P_p(.)	$c_5 * \|I\|$; I: input Termlist
TJ_{qt}(stream*)	$c_6 * \|input\| + c_7 * \|output\|$
L(s1, s2)	$c_8 * (\|s1\| + \|s2\|) + c_9 * \|output\|$
O(s)	$c_{10} * \|s\|$

By analyzing all components of the SF and KF approaches, we define a set of operators for the integrated system, as summarized in Table 1. Among the seven operators, the first two operators are classified as I/O operators, the following two operators are filters, and the last two operators are introduced because they are required during the process of the KF approach. Specifically, operators T and K identify elements based on the given tag t and keyword k, respectively. Operators C and P return elements satisfying the full-text constraint based on op, $t1$ and $t2$, and the path constraint p, respectively. Operator TJ identifies a set of *match trees* from the given streams. A match tree is a set of elements, where each component satisfies individual constraints, and the whole set satisfies the structural constraint imposed by the twig query. Finally, the *L* operator is applied to identify the LCAs which consist of the input keywords. The

O operator is required since the elements represented in an SCU table are in postorder, while the elements represented in a stream should be in preorder. This operator will perform the required order transformation.

We can apply these operators to represent different execution plans. The following expression describes a possible execution plan for *Q1* based on the SF approach, where the keywords and tag names are represented in shorthand, and qt represents the sample query tree depicted in Figure 2:

$$TJ_{qt}(P_{/c/i/r}(C^{dat,des}_{dis>=2}(C^{dat,des}_{ord}(T(remark)))),\\ P_{/c/i//n}(C^{Pet,Rob}_{ord}(T(name)))) \quad (1)$$

Specifically, this plan first processes the leaf nodes of the query tree and identifies those elements representing the given tag names. It then examines each element in the associated stream and determines if the corresponding Termlist satisfies the specified full-text predicates. Finally, it picks those elements satisfying the path constraints to form match trees and return answers.

The following expression represents another possible execution plan based on the KF approach:

$$TJ_{qt}(O(P_{/c/i/r}(C^{dat,des}_{dis>=2}(C^{dat,des}_{ord}(L(K(dat),K(des)))))),\\ O(P_{/c/i//n}(C^{Pet,Rob}_{ord}(L(K(Pet),K(Rob)))))) \quad (2)$$

Note that the *L* and *O* operators are additionally required in the KF approach.

3.2 Rewriting Rules

Table 2: Rewriting Rules

No	Rule
1	$C^{t1,t2}_{op1}(C^{t3,t4}_{op2}(.)) = C^{t3,t4}_{op2}(C^{t1,t2}_{op1}(.))$
2	$P_p(C^{t1,t2}_{op}(.)) = C^{t1,t2}_{op}(P_p(.))$
3-1	$C^{t1,t2}_{op1}(C^{t3,t4}_{op2}(L(L(s1,s2),(L(s3,s4)))) =$ $C^{t1,t2}_{op1}(C^{t3,t4}_{op2}(L(L(L(s1,s2),s3),s4))) =$
3-2	$C^{t3,t4}_{op2}(L(C^{t1,t2}_{op1}(L(s1,s2)),(L(s3,s4)))) =$
3-3	$L(C^{t1,t2}_{op1}(L(s1,s2)),C^{t3,t4}_{op2}(L(s3,s4))) =$

Table 2 lists several rewriting rules designed for this system. Rule 1 indicates that two C operators are commutable; rule 2 indicates that a C operator and a P operator are commutable. These two rules are applicable to both SF and KF approaches. There are certain rewriting rules which are only applicable in one approach. For example, rules 3-1 to 3-3 represent that the C operator can push before or after the LCA operation, which only function in the KF approach.

We then explain how to produce different execution plans. First, for each leaf node of the query tree, we derive the default-SF plan and the default-KF plan. The default-SF plan is produced by enforcing the following processing sequence: the ordered constraint, the distance constraint, and the path constraint. The default-KF plan performs the LCA operation first, and then follows the same sequence as in the default-SF plan.

After getting the default plans, as the sample execution plans (1)-(2) presented in Section 3.1, we then apply the rewriting rules to produce all possible plans. For example, rewriting rule 2 can be applied to execution plan (1) and make path filtering performed before the full-text constraint. Finally, we consider possible combinations among all leaf nodes to produce the set of complete execution plans.

3.3 The Cost Model

Table 3: Coefficients in the Cost Model

coefficient	c_1	c_2	c_3	c_4	c_5
value	12	15	1.56	1.33	4.24
coefficient	c_6	c_7	c_8	c_9	c_{10}
value	1	4	1	5	0.1

In our cost-based optimization system, we estimate the cost of each possible execution plan and choose the plan with the least cost. To do so, we design the cost model for each operator, as shown in the last column of Table 1. They are derived based on the time complexity of the corresponding algorithms, which are basically linear to the input (and/or output).[1]

The coefficients in the cost model, *i.e.*, c_1 through c_{10}, are obtained empirically based on representative datasets and queries. Briefly speaking, we divide the execution time of a query into several portions based on the operators which constitute the execution plan, and also record the amounts of data operated (and produced) within each portion. By simple calculation we get the values of coefficients. We execute each query twenty times and get the average values of coefficients. The values got from all test queries are again averaged and normalized. The final result is summarized in Table 3. In addition, we also collect several types of statistic data, which include the occurrences of each tag name and the occurrences of each keyword in an XML document, to calculate the estimated cost of an execution plan.

4. EXPERIMENT

We have designed several experiments to evaluate the performance of our integrated system. All the experiments are performed on a personal computer with Intel Core i7 3.7GHz CPU and 16GB memory, with the Microsoft Windows 7 operating system. The test queries are summarized in Table 4, where the XPath-like syntax is applied to save space. Besides, "cnt" is the shorthand

[1]We build two indices to support the two I/O operators. The first one is a hash index designed for operator T, which has the tag-name as the key, and will return the information: *(encoding, keyword, position)*. The second one is an inverted index designed for operator K, which utilizes the *keyword* as the keys, and returns the *encoding* and *position* information.

Table 4: The Test Queries

No	Query Statement
Q1	/dblp/inproceedings[./title cnt (“system”, “system”), (“advanced”, “course”), (“algorithm, “application”), (“base”, “computer”), (“information”, “image”) and ./booktitle cnt (“language”, “language”), (“conference”, “conference”), (“data”, “performance”), (“model”, “management”), (“design”, “design”)]
Q2-5	*similar to Q1 and omitted due to space limitation*
Q6	/dblp/inproceedings[./booktitle cnt (“air”, “air”) and ./title cnt (“consider”, “consider”)]
Q7	/dblp/inproceedings[./booktitle cnt (“architecture”, “architecture”) and ./title cnt (“knowledge”, “knowledge”)]
Q8	/dblp/inproceedings[./booktitle cnt (“system”, “system”) and ./title cnt (“us”, “us”)]
Q9	/dblp/inproceedings[./school cnt (“university”, “university”) and ./note cnt (“conference”, “conference”)]
Q10	/dblp/inproceedings[./editor cnt (“John”, “John”) and ./publisher cnt (“Germany”, “Germany”)]
Q11	/dblp/inproceedings[./title cnt (“System”, “System”) and ./author cnt (“Jerry”, “Jerry”)]
Q12	/site/reqions/asia/item[.//description//text//keyword cnt (“master”, “attend”, <=, 500) and .//shipping cnt (“ship”, “see”)]
Q13	/site/reqions/asia/item[.//description//text cnt (“master”, “attend”, <=, 500) and .//shipping cnt (“ship”, “see”)]
Q14	/site/reqions/asia/item[.//description cnt (“master”, “attend”, <=, 500) and .//shipping cnt (“ship”, “see”)]
Q15	/site/reqions/asia/item[.//desciption//text cnt (“master”, “attend”, <=, 5) and .//shipping cnt (“see”, “see”)]
Q16	/site/reqions/asia/item[.//desciption//text cnt (“master”, “attend”, <=, 50000) and .//shipping cnt (“see”, “see”)]
Q17	/site/reqions/asia/item[.//desciption//text cnt (“master”, “attend”, <=, 50000000000) and .//shipping cnt (“see”, “see”)]
Q18	/site//item[.//desciption//text cnt (“master”, “master”) and .//keyword cnt (“bound”, “bound”) and .//shipping cnt (“description”, “description”)]
Q19	/site//item[.//desciption//text cnt (“master”, “master”) and .//keyword cnt (“bound”, “bound”) and .//shipping cnt (“description”, “description”) .//location cnt (“Netherland”, “Netherland”)]
Q20	/site//item[.//desciption//text cnt (“master”, “master”) and .//keyword cnt (“bound”, “bound”) and .//shipping cnt (“description”, “description”) .//location cnt (“Netherland”, “Netherland”) and .//from cnt (“june”, “june”)]

of *contains text*, (A, B) represents “A ftand B ordered”,[2] and the distance constraint is represented as a quadruple. These test queries are designed to consist of a variety of cases. Q1-Q5 have the same query tree structure, but with 10, 14, 18, 22, 26 full-text constrains, respectively. Q6-Q8 also have the same query tree structure, but the constrained keywords have different frequencies, where those in Q6 have the lowest frequencies, and those in Q8 have the highest frequencies. In contrast, Q9-Q11 have different tag frequencies, where those in Q9 have the lowest frequencies, and those in Q11 have the highest frequencies. Q12-Q14 have the same full-text constraint, but the length of the root-to-leaf path decreases. The difference among Q15-Q17 is that the distance between the constrained keyword increases. Finally, Q18-Q20 have increasing numbers of path constraints.

We apply three datasets, which are two DBLP collections with the size 107MB and the size 872MB, respectively, and the XMark data set with the size 116MB. Each query is executed three times, and we calculate the average of the last two execution time.

[2]When using the same word in an ordered constraint, *e.g.*, (“system”, “system”), we only require the word to appear once.

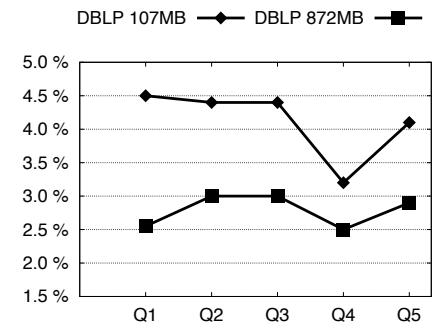


Figure 4: Overhead of Performing Optimization

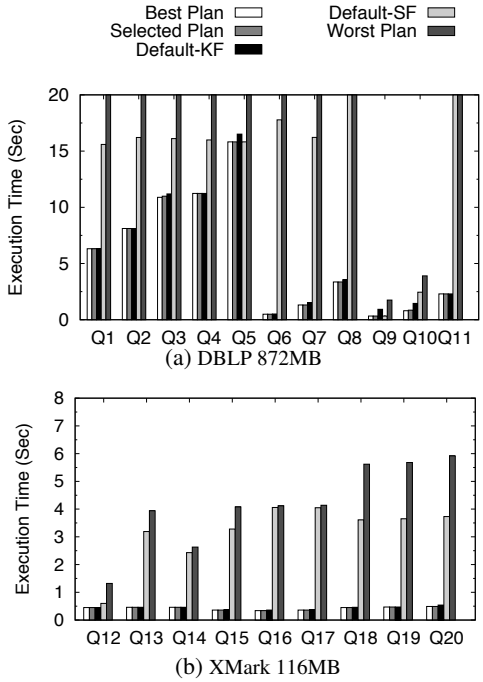


Figure 5: Effectiveness of Cost-Based Optimization

4.1 Effectiveness of Cost-based Optimization

First, we evaluate the overhead of performing optimization. Normally, when a query consists of more constraints, more alternative plans will be produced and need to be explored. However, the query might also need more time to evaluate. In Figure 4, we show the ratios of the optimization time and the execution time of selected plans. Observe that they are all below 4.5%, which are quite minor and acceptable.

Next, we evaluate the performance of our cost-based optimization system.[3] For each query, we record the execution time of the real best plan, the plan selected by our system, the default-KF plan, the default-SF plan, and the worst plan. We applied queries Q1-Q11 to the two DBLP datasets and applied queries Q12-Q20 to the

[3]Experimental results of the smaller DBLP dataset are similar to those of the larger one, and are omitted in Figures 5-6.

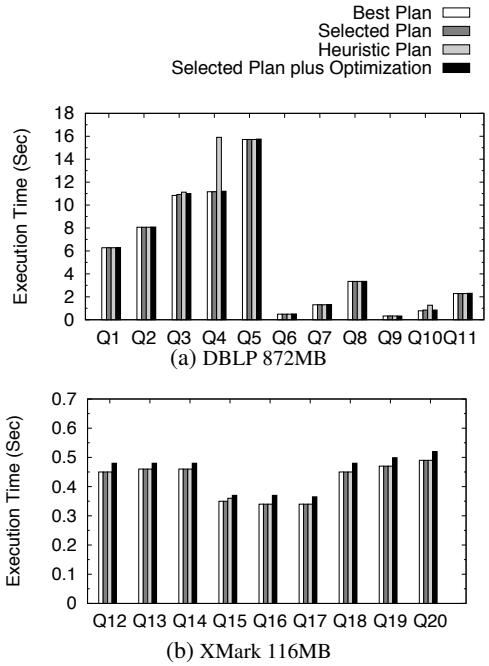


Figure 6: Effectiveness of the Heuristic Rules

XMark data set. As shown in Figure 5, for the vast majority (over 90%) of queries, our cost-based system will choose the best plan. If not, the selected plan whose execution time is all less than 8% above the cost of the actual best plan. This shows the effectiveness of our system.

Based on the execution time of each query, we observe that the KF approach usually outperforms the SF approach. The reason is that the costs of those operators comprising the KF approach and SF approach do not differ a lot. The only exception is that coefficients $c1$ and $c2$, *i.e.*, the costs of the input operations, are larger than others by an order of magnitude (See Table 3). Therefore, the amount of the input dominates the total cost. Since a document usually consists of more keywords than tags, keyword frequencies tend to be less than the tag frequencies. Therefore, it is reasonable that the KF approach is commonly more efficient than the SF approach when the query is not very complex. However, recall that the KF approach needs to do extra LCA computation and order transformation. Therefore, we can see that the KF approach will be defeated when there are many full-text constraints, as shown in Q5.

4.2 Effectiveness of the Heuristic Rules

Based on the above discussion and observing that the best plan is usually the default plan (Figure 5), we derived two heuristic rules as listed below:

1. If sum(Km) $<$ sum (Tm), choose the default-KF plan.
2. If the number of full-text constraints (N) is bigger than 20, choose the default-SF plan.

In short, our heuristic system will not apply the rewriting rules discussed in Section 3.2, but directly choose the most appropriate plan based on the following rule:

If (sum(KM) $<$ sum(Tm)) or $N < 20$ **then** choose the default-KF plan; **else** choose the default-SF plan.

As shown in Figure 6, our heuristic system works correctly in about 80% cases. When additionally considering the optimization time required by the cost-based system, our heuristic system runs faster than the cost-based system in more than 90% cases. This shows the effectiveness of our heuristic rules.

5. CONCLUSION

For an XQuery with structural and complex full-text constraints, we first design a cost-based optimizer and then derive a heuristic system to avoid performing rewriting. The experimental results show that the two approaches are both effective. In the future, we plan to explore more optimization techniques such as described in [5, 6] to further improve querying performance.

6. REFERENCES

[1] S. Amer-Yahia, E. Curtmola, and A. Deutsch. Flexible and efficient xml search with complex full-text predicates. In *Proceedings of the SIGMOD Conference*, 2006.

[2] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal xml pattern matching. In *Proceedings of the ACM SIGMOD conference*, 2002.

[3] Y.-H. Chang, C.-Y. Wu, and C.-C. Lo. Processing xml queries with structural and full-text constraints. *Journal of Information Science and Engineering*, 28(2), 2012.

[4] S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan. Twig2stack: Bottom-up processing of generalized-tree-pattern queries over xml documents. In *Proceedings of the 32rd VLDB Conference*, 2006.

[5] H. Georgiadis, M. Charalambides, and V. Vassalos. Cost based plan selection for xpath. In *Proceedings of the SIGMOD conference*, 2009.

[6] H. Georgiadis, M. Charalambides, and V. Vassalos. Efficient physical operators for cost-based xpath execution. In *Proceedings of the EDBT conference*, 2010.

[7] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *Proceedings of the SIGMOD Conference*, 2004.

[8] Z. Liu and Y. Chen. Reasoning and identifying relevant matches for xml keyword search. In *Proceedings of the 34rd VLDB Conference*, 2008.

[9] J. Lu, T. W. Ling, C.-Y. Chan, and T. Chen. From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In *Proceedings of the VLDB conference*, 2005.

[10] The World Wide Web Consortium. XQuery and XPath full text 1.0. w3c recommendation. http://www.w3.org/TR/xpath-full-text-10/, 2011.

[11] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for topx search. In *Proceedings of the VLDB Conference*, 2005.

Quality-aware Service-Oriented Data Integration: Requirements, State of the Art and Open Challenges

Schahram Dustdar Reinhard Pichler Vadim Savenkov Hong-Linh Truong

Vienna University of Technology
{dustdar,truong}@infosys.tuwien.ac.at
{pichler,savenkov}@dbai.tuwien.ac.at

ABSTRACT

With a multitude of data sources available online, data consumers might find it hard to select the best combination of sources for their needs. Aspects such as price, licensing, service and data quality play a major role in selecting data sources. We therefore advocate quality-aware data services as a natural data source model for complex data integration tasks and mash-ups. This paper focuses on requirements, state of the art, and the main research challenges on the way to the realization of such services.

1. INTRODUCTION

The problem of transferring data between and answering queries over heterogeneous information systems has been one of the key topics of database research over the past fifteen years. Various approaches have been intensively studied. Perhaps the most popular approach is data federation [9, 34], where many sources are encapsulated in a virtual global database that translates queries against the single mediating schema into queries on the sources [35, 43].In data exchange, the data is actually materialized in the target database; the data sources are no longer needed and, possibly, no longer available for query answering [24]. Peer data management advances these basic approaches, allowing many-to-many, bi-directional mappings between the data schemas of systems participating in data integration. Queries posed against one such system are normally answered by using the local database of this system and by retrieving data from other systems in the network, which in turn may also have to contact other systems [8, 14, 37]. In [28] it was shown that all these three approaches can be considered as special cases of a general information integration framework which we refer to as *data network* in this paper. Such a data network consists of autonomous systems – *data peers* – which may have both data exchange relationships and virtual mappings between each other. Moreover, data exchange constraints as well as virtual mappings may exist locally on each data peer.

The semantics of answering queries in such data networks is now well understood [14, 24, 37, 43]. Further aspects, where a lot of progress has been made in recent research, include performance issues and identifying the barrier between tractability and intractability of query answering [1, 57, 24] and important data exchange tasks [24, 30]; providing and using information on provenance [33, 32]; privacy in a data integration environment [49]; dealing with various forms of uncertainty [7, 23], and query answering in the presence of inconsistencies [2, 19].

However, the needs of many real-world applications are still by far exceeding these current developments. Consider, for instance, the case when an enterprise wants to provide some of the data accumulated in its internal data network to third parties for a fee. With recent introduction of *data marketplaces* like Windows Azure Marketplace or Infochimps, such practice is becoming routine for many companies, with millions of data sources being offered for querying already. Currently, data marketplaces aid their customers by providing a unified interface to a multitude of data sources, including support for certain query languages, online schema browsing tools, and schema documentation and licensing information in legal English. Furthermore, marketplaces offer reliable payment processing, cloud storage facilities for improved querying performance, format conversions, and greatly streamlined step-by-step processes for data publishers and data consumers. With a large number of datasets available in the marketplaces, the need for their comprehensive specification becomes especially apparent for users trying to identify the best data sources matching their requirements. As the example of conventional e-commerce platforms shows, extensive product descriptions (that is, metadata) and quality of the information products is crucial. Sim-

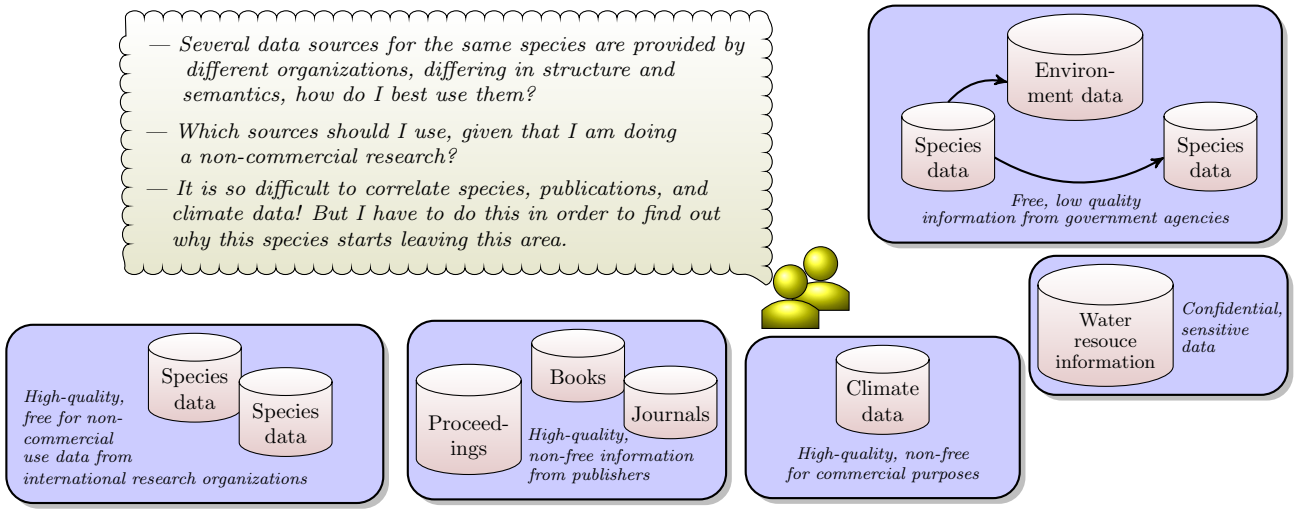


Figure 1: Challenges faced by the user when using biodiversity data for researching species

ilarly to Amazon customers checking other users' reviews when considering a purchase, customers of data marketplaces will rely on the quality metrics and community ratings of the information products offered to them. Moreovoer, mash-up applications will require most of this metadata, along with licensing and pricing terms to be machine readable. Such requirements might translate into complex measures for assessing, e.g., data quality at the data publisher side. Thus, a means for automatic generation and maintenance of metadata from a data network at hand will be greatly beneficial for both data publishers and data consumers.

In this paper we take a *quality-aware service-oriented approach* to data integration and consider Data Services (DSs) built on top of conventional databases or data networks. The main task of DSs is to provide online access to its data *and metadata* while meeting certain quality standards, known as service contracts. We believe that data quality must be a core of service contracts, along with further service-related quality criteria, data license and pricing [67]. We will discuss the challenges associated with building DSs and align those challenges with the state-of-the art in research in information integration and service-oriented architectures.

2. A CASE FOR DATA SERVICES

To elaborate possible requirements for quality-aware data services outlined in the introduction, let us consider the case of biodiversity data integration, illustrated in Figure 1. As described in [69], biodiversity data have several properties which strongly impact data integration: (i) Biodiversity data can have complex structure, describe a large number of species and observations, with broadly varying degrees of quality. (ii) Many organizations are involved in providing and continuously updating the data. Data sources of different origin vary in semantics and formats. (iii) Similar and/or complementary data is provided by different organizations. There are many data owners who can set different data access policies. (iv) Intellectual property rights (IPRs) or other disclosure concerns may apply, for example, to data acquired by commercial research.

Biodiversity data is available from the Global Biodiversity Information Facility (`www.gbif.net`). Depending on specific research goals, it may need to be enhanced, e.g., with climate or water resource information.

Assume that a user wants to determine why a specific type of species starts to disappear from a particular area. She probably needs to access and integrate the data from multiple sources scattered over the Internet (see Figure 1). These sources can be free for non-commercial purposes or completely free, and sensitive or associated with some complex IPRs. Similar data can be offered by different providers, with varying structure and quality. For the user, it can be difficult to select the optimal combination of data sources for a task at hand.

This scenario poses several research questions. To aid the user in her job, one would need to know, e.g. how to support the discovery of data services based on quality and licensing aspects, and how to support the composition of the data from different quality-aware data services?

From the data publisher point of view, the related questions are: How to choose the quality, licensing and pricing models for the data service? How to align these models with the underlying data network, and how to perform the necessary measure-

ments? Is it possible to optimize the data network (which is most likely in use for quite a few enterprise tasks) for data publishing, so that the service contracts are observed?

We will discuss some of the research issues associated with the implementation of the above scenario in the next section.

3. ENABLING DATA SERVICES

One of the most general architectural models for data integration considered in the literature is a *peer data network*, in which nodes are independent databases (data peers) and links between nodes represent schema mappings or other specifications of data dependencies. Our goal is to provide an infrastructure for publishing the data from a data network, through a thoroughly specified interface, which we have introduced before as *data service*. In this section, we subdivide such an infrastructure into three conceptual levels:

- *Data Network Level*, concerned with the data network management tasks, such as querying and updating data and metadata (e.g., data lineage) and collecting the quality and performance metrics.
- *Service Interface Level*, concerned with the presentation of the distributed enterprise data to the user through a single service interface, including formal specification of the service, data and quality models.
- *Meta-Service Level*, performing discovery and integration of multiple services (possibly, other data services).

The set of specific issues that would arise in practice is largely determined by such factors as the data model used by data peers, expressiveness of mappings and ETL operations used to transfer the data, the nature and volume of the published information, required query capabilities, and countless further considerations. However, as we point out in the remainder of this section, already catering for a basic data service functionality with relational databases as peers leads to a number of open issues and research opportunities.

3.1 Managing metadata in data networks

To enable the data service interface, the data network must be able to to store, maintain, and provide access to the metadata describing the data quality, licensing and pricing properties. This functionality is by no means straightforward, as we will argue shortly. For example, during updates or caching, both data and metadata changes must be propagated through the network. For many classes of metadata, already the basic rules of propagation through the schema mappings have to be defined. For instance, what does it mean to propagate the data quality descriptors. Even a comparatively simple completeness metric depends crucially on the semantics of the schema mappings, namely if the closed- or open-world assumption is taken [60]. Here, we discuss such questions for several types of metadata relevant for the functionality of data services.

Provenance, often also called "lineage", is the key type of metadata on which the derivation of other metrics in data networks depends. Usually, one distinguishes the "why-" and "where-" provenance [13], where the former gives a comprehensive justification for creation of a data item, while the latter references the origins of the values used in a derived data item. A unified way of treatment of annotations propagated through data dependencies was presented in [33], with the examples of handling incompleteness, probabilities, why-provenance, and mulitplicities of tuples. No similar work considering the unified treatment of metadata relevant for data services (i.e., related to quality, licensing or pricing) is known to us.

Data quality (DQ) was identified as an intrinsic issue for information systems in the early 90-s [63]. Since then, it has become a subject of a huge body of publications (see [5] for a survey). The main problem addressed in research is the construction of frameworks and methodologies for assessment and continuous improvements of the enterprise DQ. For data services, the following issues play a crucial role: (i) measurement and propagation of DQ through the data network, (ii) extraction of a concise characterization of DQ as part of the data service description, (iii) quality-aware query answering, and (iv) maintenance of the DQ model associated with the data network (e.g., accommodating updates).

The typical DQ dimensions considered by the majority of authors are *accuracy, completeness, timeliness* and *consistency* [55, 5]. Each dimension can be represented by one or several quality metrics. The metrics and further quality dimensions heavily depend on the application domain, or *context*: the quality of data is often identified with its "fitness for use" [59]. Formalizing the notion of context in data quality is an interesting challenge, which only recently received attention [10]. However, even within a fixed context, assessing data quality is commonly recognized as a highly non-trivial problem, often requiring human intervention (in the form of expert evaluation or rating by the community of data users [54]). Such aspects as heterogeneity of the data add to the intricacy of the issue [4]. Still, for some qual-

ity metrics, unsupervised measurement algorithms have been studied in the literature [3, 29] and approved in practice [46]. This line of research and industrial usage, supported by tools from the major vendors of enterprise data management software, has gained a lot of momentum in recent years.

For data services offering *information products* composed from disparate enterprise data, an ability to *derive* the quality metrics of the product based on the quality metrics of the data sources is desired, instead of performing the assessment anew. The foundation for such a derivation has been laid by Wang et al. in [72], where computing the DQ metrics of the basic SQL operators' results is considered, and further extended by Sarkar et al. [51, 52]. For data networks, with multiple data flows and expressive schema mappings between the peers, the problem of deriving DQ metrics becomes increasingly complex, leaving an ample space for future research.

One of the dimensions often identified with data quality is the existence of object duplicates, which do not agree on all attribute values. Deduplication, or *record linkage*, is especially important for cases where data is distributed over several sources (which is our assumption in this paper). In a simple case, duplicates may result in violations of primary key constraints (and then correlate with the consistency DQ dimension). This case is often studied in the database literature, under the tile *consistent query answering*. For an inconsistent database state, a minimal set of updates (a minimal *repair*) is constructed, and *certain answers* – present in the query results under each repair – are computed.

However, there may be cases where conflicting duplicates do not lead to constraint violations: for instance, if inaccurate key values are present. Moreover, defining repairs through simple operations of, e.g. tuple deletion can lead to undesirable information loss. More sophisticated and domain-dependent record linkage techniques are then used, giving rise to the *data fusion* process [11]. Object linkage algorithms are often complex and can hardly be captured by simple logical formalisms like the language of database constraints.

As far as *quality-driven querying* of data is concerned, the main reference so far remains the work of Naumann [50], who proposed a framework for taking quality-related metrics into account when answering queries against web sources (i.e., in a data federation setting). More recent advances on quality-aware query answering can be found in [10, 6]. The quality metadata in XML are considered in [61, 47], where a decentralized architecture of "quality brokers", or quality-aware query mediators, is proposed. Such an architecture is well suited for data networks. However, so far only the simplest mappings between the sources and the global schema have been considered [50, 61]. Taking more expressive mappings into account, including those between data peers, can be seen as a reasonable next goal for quality-aware data querying.

The study of maintenance of data quality metrics has been merely initiated [45]. Here again, the data federation model rather than a data network with complex patterns of update propagation is considered (see also Section 3.3).

Licensing metadata, to the best of our knowledge, has received only little attention in database research. One of the most important developments is the Open Data license [48], which allows one to license a published dataset under various conditions. An overview of further real world data contracts including licensing terms can be found in [67]. However, such questions as combining the differently licensed data in the same information product and checking for licensing conflicts still wait for further exploration.

Pricing in conjunction with query processing is a typical instrument in query optimization: all resources needed to evaluate the query are generalized as monetary cost (per byte transferred, per disk read operation etc.). A total cost of a query plan is then computed. Many cloud services nowadays implement such cost models literally by charging users for actual resource consumption. Such approaches are not the ultimate answer for data services though, since this kind of pricing is content- and quality-independent. Query optimization under per-tuple fees or fees depending on DQ can be seen as viable directions for research. For instance, one can ask for best quality-dependent query plans for a given price range, or optimize data pricing for a set of typical user queries.

Quality of service (QoS) describes the properties of the data service, which are not specific for the content it serves. Typical QoS aspects are reliability, availability, security, and performance. These properties are an important part of the data service specification as well, and have to be aligned with capabilities of the underlying data network. For instance, the reliability and availability of the data service depend on the queries that can be answered at a given time instant, which in turn depends on the availability of data peers.

For data service discovery and selection (see Section 3.4) the underlying data network must be able to characterize its ability to support data services in

terms of a schema or ontology of the data as well as the above mentioned quality and licensing criteria. Such a characterization has to be given in principle (if the design of the data network supports a specified service) as well as for the current state of the network characterized by quality metrics (e.g., logical consistency and data availability).

Data network models are typically built using logical formalisms [28, 14, 37]. To support data services, such models must take metadata as part of their basic concepts: For example, consistency of sources equally depends on the data and on the metadata (e.g., sources might disagree on the license of a data item). Moreover, the sources must adhere to the same definition of quality metrics.

Further extensions of the foundations are needed as well. One extension concerns the type of mappings between the data peers. In contrast to distributed databases, there is an independent application behind each data peer, employing processing mechanisms that are generally hard to formalize declaratively. Therefore, the notion of uncertain mappings (based on [23]) with possible adaptations has to be studied to approximate the data processing rules. Such mappings are important both as a means of expressing the dependencies between the local and global schemas, but also as a way of simulating the application logic at the data peers. Another issue is concerned with the data format heterogeneity in the data network: The data with varying levels of structure (as addressed in [36]) could be integrated (in particular, semi-structured and even unstructured data) provided that the underlying data peers are able to formulate and support a certain level of data and service quality.

3.2 Publishing a data service

A data service must provide a comprehensive specification of the information products it offers. This specification will then allow data consumers to make informed decisions when searching for and comparing data sources for their applications.

Currently, QoS models for Web services are well developed and various techniques and tools support engineers modeling QoS for Web services [42, 58, 73]. However, the data-specific aspects of service description have not yet received proper support. In [66], various data-related aspects of service specifications – called *data concerns* – are identified and analyzed, and then in [65, 71] a service engineering process for publishing those concerns is developed.

Given the considerable effort spent on data quality from the database perspective, as outlined in the previous section, there is a lack of integration between DQ metrics and parameters commonly used to specify data services. In fact, no standard model of data quality that could serve as a basis for the data service specification is available so far. Furthermore, since the data quality notions are often domain-specific, actually a hierarchy of quality models will be desirable, whereby the general quality metrics like completeness can be extended by or mapped to the specialized ones (like "number of OCR failures" for an electronic library), allowing the customer to better compare similar services and still have compatible service descriptions for the case of data mash-up. Similarly, existing service licensing [26] and service level agreements (SLAs), see e.g, [39], are mainly based on "operational" QoS models. Some data licensing models exist [21], but are neither standard nor formalized to allow automatic processing, and thus cannot be used in the service model. Typical *data rights* that license models have to support include derivation, collection, reproduction, attribution and restriction to noncommercial use [67].

To date, these types of metadata have not been combined in a single service model [71], and there exist no specifications to support the publishing and discovery of data services based on such information. This calls for a new research focus on the development of publishing and discovery of quality aspects of data services. The data publishing not only requires an appropriate description of the semantics of the data (e.g. a schema or an ontology), but also a more general specification reflecting the data quality, QoS, and data service licensing. In addition to languages and specifications for modeling these types of information together, we also need a scalable and extensible framework to manage the lifecycle of this information. Existing SOA techniques can address many aspects in the modelling, publishing and management of data services. In our view, existing QoS and service licensing models can be extended with data quality metrics. Then, quality-related metadata can be linked with other types of service information, for instance, integrated into the Web services information model [64].

3.3 Optimizing the design and operation of the data network

To provide a certain quality of service with minimal cost, various kinds of optimization – both at design time and at run time – are required. This leads to the following issues.

The first issue is *network optimization*. Network topologies are characterized by the data stored on each data peer and the inter-peer mappings. Hence, the optimality of a network topology and also the equivalence of several network topologies depend on

the quality parameters of the service. Various kinds of optimization of the design of a data network have to be considered – taking the formalization of QoS in a data network into account. The concrete values of the QoS metrics are influenced by the design of the network topology, which heavily depends on data aspects like data allocation, data replication, and inter-peer mappings. Thus, a number of optimization problems naturally arises from the QoS metrics like, e.g., minimizing the cost of the data network while guaranteeing a certain QoS level. Another interesting research question concerns equivalence between network topologies (understood, e.g., as a set of schema mappings between peers, together with local data constraints), with respect to certain quality metrics. This is in spirit of the recent work by Fagin et al. [25], where several notions of equivalence between schema mappings have been proposed, in particular, relative to a given class of queries. A study of replication in data networks [62], but taking QoS into account is also a promising research direction.

Second, *query optimization* is an important issue in all data management systems – no matter whether they are considered in the context of data services or not. In a data network, traditional approaches to query optimization (see, e.g., [31, 41]) have to be extended so as to take the inter-peer and intra-peer mappings into account. Two important sub-problems that deserve further study are query routing and load distribution (see, e.g., [56]).

Finally, the *optimization of updates*, i.e., update, insert, or delete operations in a data network is an important issue. The modification of data at one peer normally leads to the violation of the inter- and intra-peer mappings, which are re-enforced by performing the chase procedure. Update optimization in a data network has two important facets. First, minimizing the propagation time or minimizing the impact on the other data peers is critical. Second, there may exist many possible states of the data network such that all constraints are fulfilled again after the update. In this case, an important optimization problem is concerned with finding the "smallest" one, which is usually referred to as the core. In [28], core computation has been extended from data exchange to a data network. In addition, core computation methods should be made incremental, i.e., be able to "locally repair" the core after an update, rather than recompute it from scratch.

Updating the data network brings about the problem of transaction support. A discussion of transactions in a service-oriented distributed environments can be found in [68]. In the context of data services, the mapping from the inter-service transaction semantics to the underlying data network transactions must be investigated.

3.4 Selecting and mixing data from quality-aware data services

With published quality- and licensing-related metadata, the data consumers will have a chance to decide if the data from a given data service fits their intended usage scenario. This is in clear contrast with a situation where only descriptions of syntax (format) and semantics (ontology) of the data are available.

Rich data service specifications are especially important when discovery, comparison, and selection of data sources are computer assisted. Users should be able to automatize such tasks as comparing and checking compatibility of data service contracts, or determining trade-offs between quality, licenses and costs associated with information products from a certain set of data services.

To support the composition of data sources on the Internet, in particular in the collaborative Web 2.0 context, many tools have been developed [22, 38]. However, existing techniques mainly focus on selecting data sources based on data structures and on dealing with syntax and semantics of the data [22, 40, 53]. In the area of Web data mash-ups, a need of considering data quality has been understood [17], and the study of DQ models initiated [16]. More comprehensive data mash-up models, including licensing and pricing in conjunction with data and service quality, are yet to be developed.

Similarly, most of the contemporary service selection and combination techniques are built around the QoS, cost, and the semantics of service operations [58, 73], rather than content-related aspects. This is true, in particular, for concepts such as ad-hoc flows [70]. For service mash-ups [44], a need for data quality support has been recently realized. In particular, the Mashup Services System [12] takes information quality metrics into account when generating and managing service mash-ups.

Service selection techniques currently do not deal with the compatibility between different license models [27] when integrating data from different services. A recent work has supported the evaluation of service contracts, but its support of data-related concerns is limited [20].

Hence, we envisage a proliferation of new techniques and algorithms for composition of data services (be it a mash-up Web application for end users, or a new data service) taking the full spectrum of data concerns and QoS aspects into account. Such algorithms and techniques should extend current

service contract compatibility evaluation [20], syntax mismatching [40], and QoS-aware and semantic workflow composition [18, 15] with data quality and contract aspects. Moreover, unlike current techniques which require user interaction for selection and maping of the data sources [53], mixing data services should be possible with minimum involvement (or no involvement at all) of the end user.

4. CONCLUSION

We have presented the concept of data services which takes a quality-aware service-oriented view on data integration, and identified the main research challenges on the way to its realization.

The main benefit of this approach comes from the cross-fertilization of the database and distributed systems fields: The peer data management concept [8, 14, 28, 37] was a first step in this direction by making P2P technology available to data management and vice versa. We consider combining service-oriented computing techniques with database technology as an important second step. This will lead to a better understanding and interesting extensions of the frameworks and methods on both sides: On the one hand, SOA key concepts like quality, licensing, service selection, and service composition must be extended so as to take data aspects into account. On the other hand, the concept of data networks must be significantly enhanced by integrating quality considerations into it and by developing new methods of data network optimization with respect to the various quality criteria.

Acknowledgements. This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-032. Savenkov was supported by the Erasmus Mundus External Co-operation Window Programme of the European Union.

5. REFERENCES

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. ACM PODS '98*, pages 254–263.

[2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proc. IEEE ICDE '06*.

[3] D. P. Ballou, I. N. Chengalur-Smith, and R. Y. Wang. Sample-based quality estimation of query results in relational database environments. *IEEE Trans. Knowl. Data Eng.*, 18(5):639–650, 2006.

[4] C. Batini, D. Barone, F. Cabitza, and S. Grega. A Data Quality Methodology for Heterogeneous Data. *Int. J. of Database Management Syst.*, 3(1):60–79, Feb. 2011.

[5] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, 2009.

[6] D. Beneventano and R. C. N. Mbinkeu. Quality-Driven Query Processing Techniques in the MOMIS Integration System Advances in Databases and Information Systems. In *Proc. ADBIS '10*, pages 46–57. Springer, 2011.

[7] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[8] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. WebDB '02*, pages 89–94, 2002.

[9] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, 2008.

[10] L. E. Bertossi, F. Rizzolo, and L. Jiang. Data quality is context dependent. In *Proc. BIRTE '10*, pages 52–67. Springer, 2011.

[11] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, jan 2009.

[12] A. Bouguettaya, S. Nepal, W. Sherchan, X. Zhou, J. Wu, S. Chen, D. Liu, L. Li, H. Wang, and X. Liu. End-to-end service support for mashups. *IEEE Trans. Serv. Comput.*, 3:250–263, July 2010.

[13] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. ICDT '01*, pages 316–330. Springer, 2001.

[14] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. In *Proc. ACM PODS '04*, pages 241–251.

[15] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proc. GECCO '05*, pages 1069–1075. ACM, 2005.

[16] C. Cappiello, F. Daniel, A. Koschmider, M. Matera, and M. Picozzi. A quality model for mashups. In *Proc. ICWE '11*, pages 137–151. Springer, 2011.

[17] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso. Information quality in mashups. *IEEE Internet Computing*, 14(4):14–22, 2010.

[18] J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 21(3):191–225, 2003.

[19] J. Chomicki. Consistent query answering: The first ten years. In *Proc. SUM '08*, pages 1–3. Springer, 2008.
[20] M. Comerio, H.-L. Truong, F. D. Paoli, and S. Dustdar. Evaluating contract compatibility for service composition in the $SeCO_2$ framework. In *Proc. ICSOC/ServiceWave '09*, pages 221–236. Springer, 2009.
[21] Committee on Licensing Geographic Data and Services, National Research Council. *Licensing Geographic Data and Services*. The National Academies Press, 2004.
[22] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah. Data integration in mashups. *SIGMOD Record*, 38(1):59–66, 2009.
[23] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proc. VLDB '07*, pages 687–698. ACM, 2007.
[24] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
[25] R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a theory of schema-mapping optimization. In *Proc. ACM PODS '08*, pages 33–42.
[26] G. R. Gangadharan and V. D'Andrea. Licensing services: Formal analysis and implementation. In *Proc. ICSOC '06*, pages 365–377. Springer, 2006.
[27] G. R. Gangadharan, H. L. Truong, M. Treiber, V. D'Andrea, S. Dustdar, R. Iannella, and M. Weiss. Consumer-specified service license selection and composition. In *Proc. IEEE ICCBSS '08*, pages 194–203.
[28] G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *Proc. ACM PODS '07*, pages 133–142.
[29] L. Golab, F. Korn, and D. Srivastava. Discovering pattern tableaux for data quality analysis: a case study. In *QDB '11*.
[30] G. Gottlob and A. Nash. Efficient core computation in data exchange. *J. ACM*, 55(2):9:1–9:49, 2008.
[31] G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.
[32] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *Proc. VLDB '07*, pages 675–686. ACM, 2007.
[33] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. ACM PODS '07*, pages 31–40.
[34] L. M. Haas. Beauty and the beast: The theory and practice of information integration. In *Proc. ICDT '07*, pages 28–43. Springer, 2007.
[35] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
[36] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *Proc. ACM PODS '06*, pages 1–9.
[37] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *IEEE ICDE '03*, pages 505–516.
[38] V. Hoyer and M. Fischer. Market overview of enterprise mashup tools. In *Proc. ICSOC '08*, pages 708–721. Springer, 2008.
[39] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *J. Network and Systems Management*, 11(1):57–81, 2003.
[40] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Trans. Serv. Comp.*, 2(2):94–107, 2009.
[41] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.
[42] K. Lee, J. Jeon, W. Lee, S.-H. Jeong, and S.-W. P. (eds.). QoS for web services: Requirements and possible approaches, Nov. 2003. W3C Technical Report.
[43] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. ACM PODS '02*, pages 233–246.
[44] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards service composition based on mashup. In *Proc. IEEE SCW '07*, pages 332–339.
[45] A. Marotta. Managing source quality changes in a data integration system. In *Proc. CAiSE '06 Doct. Consortium*. CEUR-WS.org, 2006.
[46] E. Masuoka, D. Roy, R. Wolfe, J. Morisette, S. Sinno, M. Teague, N. Saleous, S. Devadiga, C. O. Justice, and J. Nickeson. MODIS land data products: Generation, quality assurance and validation land remote sensing and global environmental change. volume 11 of *Remote Sensing and Digital Image Processing*, chapter 22, pages 509–531. Springer, 2011.
[47] D. Milano, M. Scannapieco, and T. Catarci. A peer-to-peer service supporting data quality: Design and implementation issues. In *Proc. ICSNW '04*, pages 321–322. Springer, 2004.

[48] P. Miller, R. Styles, and T. Heath. Open data commons, a license for open data. In *Proc. LDOW '08*. CEUR-WS.org, 2008.
[49] A. Nash and A. Deutsch. Privacy in GLAV information integration. In *Proc. ICDT '07*, pages 89–103. Springer, 2007.
[50] F. Naumann. *Quality-Driven Query Answering for Integrated Information Systems*. Springer, 2002.
[51] A. Parssian, S. Sarkar, and V. S. Jacob. Assessing Data Quality for Information Products: Impact of Selection, Projection, and Cartesian Product. *Management Science*, 50(7):967–982, 2004.
[52] A. Parssian, S. Sarkar, and V. S. Jacob. Impact of the Union and Difference Operations on the Quality of Information Products. *Inf. Syst. Research*, 20(1):99–120, 2009.
[53] D. L. Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proc. WWW '09*. ACM, 2009.
[54] R. Pichler, V. Savenkov, S. Skritek, and H. L. Truong. Uncertain databases in collaborative data management. In *Proc. MUD '10*, CTIT Workshop Proc. Series. Univ. Twente, 2010.
[55] L. L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Commun. ACM*, 45:211–218, 2002.
[56] T. Pitoura and P. Triantafillou. Load distribution fairness in P2P data management systems. In *IEEE ICDE '07*, pages 396–405.
[57] R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.
[58] S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.
[59] T. C. Redman. Second-generation data quality systems, chapter 34 of *Juran's quality handbook*, J.M. Juran and G.A.Blanton, eds. 1999.
[60] M. Scannapieco and C. Batini. Completeness in the relational model: a comprehensive framework. In *Proc. IQ '04*, pages 333–345. MIT, 2004.
[61] M. Scannapieco, A. Virgillito, C. Marchetti, M. Mecella, and R. Baldoni. The DaQuinCIS architecture: a platform for exchanging and improving data quality in cooperative information systems. *Inf. Syst.*, 29(7):551–582, 2004.
[62] M. Sozio, T. Neumann, and G. Weikum. Near-optimal dynamic replication in unstructured peer-to-peer networks. In *Proc. ACM PODS '08*, pages 281–290.
[63] D. M. Strong, S. E. Madnick, T. Redman, A. Segev, and R. Y. Wang. Data quality: A critical research issue for the 1990s and beyond. In *Proc. ICIS '94*, pages 500–501. Assoc. for Inf. Systems, 2004.
[64] M. Treiber, H.-L. Truong, and S. Dustdar. Semf — service evolution management framework. In *Special session on Quality and Service-Oriented Applications, SEAA '08*, pages 329–336. IEEE, 2008.
[65] H. L. Truong and S. Dustdar. On evaluating and publishing data concerns for data as a service. In *Proc. IEEE APSCC '10*, pages 363–370.
[66] H.-L. Truong and S. Dustdar. On analyzing and specifying concerns for data as a service. In *Proc. IEEE APSCC '09*, pages 87–94, 2009.
[67] H.-L. Truong, G. Gangadharan, M. Comerio, S. Dustdar, and F. D. Paoli. On analyzing and developing data contracts in cloud-based data marketplaces. In *Proc. IEEE APSCC '11*, pages 174–181, 2011.
[68] C. Türker, K. Haller, C. Schuler, and H.-J. Schek. How can we support grid transactions? towards peer-to-peer transaction processing. In *Proc. CIDR '05*, pages 174–185.
[69] O. Unal and H. Afsarmanesh. A final report on distributed information management requirement analysis. Technical report, Uiv. Amsterdam, Inf. Dept., 2004.
[70] M. Voorhoeve and W. M. P. van der Aalst. Ad-hoc workflow: Problems and solutions. In *Proc. DEXA Workshop '97*, pages 36–40. IEEE Comp. Soc. Press, 1997.
[71] Q. H. Vu, T.-V. Pham, H.-L. Truong, S. Dustdar, and R. Asal. On analyzing and developing data contracts in cloud-based data marketplaces. In *Proc. IEEE AINA '12*. to appear.
[72] R. Y. Wang, M. Ready, and H. B. Kon. Toward quality data: An attribute-based approach. *Decision Support Systems*, 13:349–372, 1995.
[73] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-aware selection model for semantic web services. In *Proc. ICSOC '06*, pages 390–401. Springer, 2006.

A Survey of View Selection Methods

Imene Mami and Zohra Bellahsene
Université Montpellier 2 - INRIA, LIRMM, Montpellier, France
{imen.mami,bella@lirmm.fr}

ABSTRACT

Materialized view selection is a critical problem in many applications such as query processing, data warehousing, distributed and semantic web databases, etc. We refer to the problem of selecting an appropriate set of materialized views as the view selection problem. Many different view selection methods have been proposed in the literature to address this issue. The present paper provides a survey of view selection methods. It defines a framework for highlighting the view selection problem by identifying the main dimensions that are the basis in the classification of view selection methods. Based on this classification, this study reviews most of the view selection methods by identifying respective potentials and limits.

1. INTRODUCTION

The view selection issue has been investigated in several contexts: query optimization, warehouse design, data placement in a distributed setting, web databases, etc. Many diverse solutions to the view selection problem have been proposed and analyzed through surveys [13, 21, 32]. The survey [21] concentrates on methods of finding a rewriting of a query using a set of materialized views. The study presented in [32] focuses on the state of the art in materialization for web databases. A critical analysis of methodologies for selecting materialized views in data warehousing is provided in [13]. However, none of the above mentioned surveys provides a classification of view selection approaches in order to identify their advantages and disadvantages. Our survey fills this gap.

This paper aims at studying the view selection in relational databases and data warehouses as well as in a distributed setting. First, we define a framework for highlighting the view selection problem. Thus, we present a classification of view selection methods based on the main view selection dimensions that we have identified. This study also reviews existing view selection methods by identifying respective potentials and limits.

The rest of the paper is organized as follows: Section 2 gives some definitions. Section 3 identifies the main view selection dimensions along which view selection methods can be classified. Section 4 presents a critical survey of existing view selection methods. Section 5 contains the conclusion and discusses open issues.

2. PROBLEM SPECIFICATION

2.1 Preliminaries

Here, we introduce the main notions used in this paper and related to the view selection context.

View: A view is a derived relation, defined by a query in terms of base relations and/or other views.

Materialized View: A view is said to be materialized if its query result is persistently stored otherwise it is said to be virtual. We refer to a set of selected views to materialize as a set of materialized views.

Workload: A workload or a query workload is a given set of queries $Q = \{Q_1, Q_2, \cdots, Q_q\}$. Each query Q_i has an associated non-negative weight fQ_i which describes the query frequency. The set of materialized views is dependent on the query workload. In a distributed scenario, the queries are executed on different computer nodes. Each computer node has an associated query workload.

View Selection: Given a database schema and a query workload, the objective is to select an appropriate set of materialized views to improve query performance. The process of selecting a set of materialized views is known as view selection.

View Maintenance: Whenever a base relation is changed, the materialized views built on it have to be updated in order to compute up-to-date query results. The process of updating a materialized view is known as view maintenance. Different maintenance policies (deferred or immediate) and maintenance strategies (incremental or rematerialization)

can be applied [17, 18, 38, 58].

2.2 Problem Definition

The use of materialized views is a common technique to reduce query response time [6]. Indeed, materializing an appropriate set of views and answering queries using these views can significantly speed up the query processing since the access to materialized views can be much faster than recomputing the views. Therefore, materializing all the input queries can achieve the lowest query processing cost but the highest view maintenance cost since materialized views have to be maintained in order to keep them consistent with the data at sources. Besides, the query result can be too large to fit in the available storage space. Hence, there is a need for selecting a set of views to materialize by taking into account three important parameters: query processing cost, view maintenance cost and storage space. The problem of choosing which views to materialize which have a desirable balance among the three costs is known as the view selection problem. This is one of the most challenging problems in data warehousing [50] and it is known to be a NP-complete problem [26]. In a distributed environment consisting of many heterogeneous nodes with different resource constraints, the distributed view selection problem is that which view has to be materialized at which node of the network. The view selection problem in a distributed case is much more difficult than the view selection problem in a central case because of the immense challenges associated to distributed settings [16] (i.e., data granularity, degrees of replication, heterogeneity of information sources, etc.).

Problem Formulation: The problem of view selection can be formulated as follows. Given a database schema $R = \{R_1, R_2, \cdots, R_r\}$, a query workload $Q = \{Q_1, Q_2, \cdots, Q_q\}$ defined over R, the problem is to select an appropriate set of materialized views $M = \{V_1, V_2, \cdots, V_m\}$ such that the query workload is answered with the lowest cost under a limited amount of resources, e.g., storage space and/or view maintenance cost.

The view selection problem in a distributed context consisting of a set of nodes $N = \{N_1, N_2, \cdots, N_n\}$ in which each node has an associated query workload, is to choose a set of views $M = \{V_1, V_2, \cdots, V_m\}$ and a set of nodes $N_v \subseteq N$ at which M should be materialized. The distributed view selection is designed so that the full query workload is answered with the lowest cost subject to resource constraints. Resources may be storage space per node, view maintenance cost, communication costs or a combination of them.

2.3 Cost Model

The cost model is an important issue for the view selection process [9]. The main objective in view selection problem is the minimization of the weighted query processing cost, defined by the formula:

$$QueryProcessingCost = \sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M)$$

where f_{Q_i} is the query frequency of the query Q_i and $Qc(Q_i, M)$ is the processing cost corresponding to Q_i given a set of materialized views M.

Because materialized views have to be kept up to date, the view maintenance cost has to be considered. This cost is weighted by the update frequency indicating the frequency of updating materialized views. The view maintenance cost is computed as follows:

$$ViewMaintenanceCost = \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)$$

where $f_u(V_i)$ is the update frequency of the view V_i and $Mc(V_i, M)$ is the maintenance cost of V_i given a set of materialized views M.

The cost model is extended for distributed setting by taking into account the communication cost which is the cost for transferring data from its origin to the node that initiated the query. Given a query Q_i which is asked at a node N_j and denoting by V_k a view used to answer Q_i, the communication cost is zero if V_k is materialized at N_j. Otherwise, let N_l be the node containing V_k, then the communication cost for transferring V_k from N_l to N_j is:

$$CommunicationCost_{(V_k, N_l \rightarrow N_j)} = C_{N_j, N_l} * size(V_k)$$

where C_{N_j, N_l} is the network transmission cost per unit of data transferred between N_j and N_l and $size(V_k)$ is the size of the view V_k.

2.4 Static View Selection vs. Dynamic View Selection

A static view selection approach is based on a given workload and chooses accordingly the set of views to materialize. Whereas, in a dynamic view selection approach, the view selection is applied as a query arrives. Therefore, the workload is built incrementally and changes over time. Because the view selection has to be in synchronization with the workload, any change to the workload should be reflected to the view selection as well. Indeed, in a system of a dynamic nature [4, 5, 29], the set of

materialized views can be changed over time and replaced with more beneficial views in case of changing the query workload. In order to reduce view maintenance cost and storage space requirements, [57] aims at materializing the most frequently accessed tuples of the view rather than materializing all tuples of the view. The set of materialized tuples can be changed dynamically as the queries change, either manually or automatically by an internal cache manager using a feedback loop. However, the task of monitoring constantly the query pattern and periodically recalibrating the materialized views is rather complicated and time consuming especially in large data warehouse where many users with different profiles submit their queries.

A dynamic view selection is often referred to as view caching. With caching, the cache is initially empty and data are inserted or deleted from the cache during the query processing. Materialization could be performed even if no queries have been processed and materialized views have to be updated in response of changes on the base relations. A detailed comparison of these two techniques is given in [27]. Traditional caching approaches aim at caching the results of queries, in other words to cache views. Another alternative is to cache only a part of a view. Indeed, a chunk based scheme has been introduced in [12] for fine granularity caching. Chunk based caching allows caching of only few, frequently used tuples of views. To facilitate the computation of chunks required by a query but not found in the cache, a new organization for base relations has been proposed which they called a chunked file. Caching has been adopted in data warehousing [43], distributed databases [28] and peer to peer systems [25]. Dynamic view indexing has also been considered in [44]. In this paper, we focus only on static view selection methods because most of existing view selection approaches are of static nature.

3. VIEW SELECTION DIMENSIONS

In order to identify the advantages and disadvantages of view selection methods, we propose two main dimensions along with they can be classified: (i) Frameworks; and (ii) Resource Constraints.

3.1 Frameworks

Generally, approaches to the view selection problem consist of two main steps. The first step identifies the candidate views which are promising for materialization. Techniques based on multiquery DAG, syntactical analysis of the workload or query rewriting have been used to obtain the candidate views. Based on the set of candidate views, the second step selects the set of views to materialize under the resource constraints and by applying heuristic algorithms.

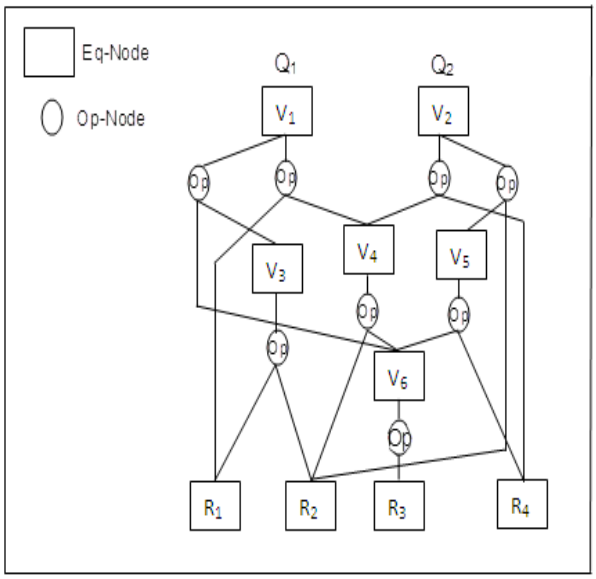


Figure 1: The AND-OR view graph of the two queries Q_1 and Q_2.

3.1.1 *Multiquery DAG*

Most of the proposed view selection methods operate on query execution plans. The plans can be derived from multiple query optimization techniques or by merging multiple query plans. The main interest of such techniques relies in detecting common sub-expressions between the different queries of workload and capturing the dependencies among them. This feature can be exploited for sharing updates and storage space. The dependence relation on queries (or views) has been represented by using a directed acyclic graph also called a DAG. However, these methods require optimizer calls which can be expensive in complex scenarios.

The most commonly used DAGs in literature are:

AND/OR View Graph: The union of all possible execution plans of each query forms an AND-OR view graph [40]. The AND-OR view graph described by Roy [42] is a Directed Acyclic Graph (DAG) composed of two types of nodes: Operation nodes and Equivalence nodes. Each operation node represents an algebraic expression (Select-Project-Join) with possible aggregate function. An equivalence node represents a set of logical expressions that are equivalent (i.e., that yield the same result). The operation nodes have only equivalence nodes as

children and equivalence nodes have only operation nodes as children. The root nodes are the query results and the leaf nodes represent the base relations. A sample AND-OR view graph is shown in figure 1. Circles represent operation nodes (Op-Nodes) and boxes represent equivalence nodes (Eq-Nodes). For example, in figure 1, view V_1 corresponding to a single query Q_1, can be computed from V_6 and V_3 or R_1 and V_4. If there is only one way to answer or update a given query, the graph becomes an AND view graph. In the data cube which is a specific model of a data warehouse, the AND-OR view graph is an OR view graph, as for each view there are zero or more ways to construct it from other views, but each way involves only one other view [19]. In other words, an OR view graph is an AND-OR view graph in which any node is an equivalence node that can be computed from any one of its children.

Multi-View Processing Plan (MVPP): The MVPP defined by Yang et al [52] is a directed acyclic graph in which the root nodes are the queries, the leaf nodes are the base relations and all other intermediate nodes are selection, projection, join or aggregation views that contribute to the construction of a given query. The MVPP is obtained after merging into a single plan either individual optimal query plans (similar to the AND view graph) or all possible plans for each query (similar to the AND-OR view graph). The difference between the MVPP representation and the AND-OR view graph or the AND view graph representation is that all intermediate nodes in the MVPP represent operation nodes. A sample MVPP is shown in figure 2.

Data Cube Lattice: Harinarayan and al [22] propose modeling data in multiple dimensions. It is built from the queries involved in the data warehouse application, e.g., OLAP-style queries. The Data Cube Lattice is a DAG whose nodes represent queries (or views) which are characterized by the attributes of the Group by clause. The edges denote the derivability relation between views. That is, if there is a path from view V_i to a view V_j (see figure 3), then grouping attributes on V_j can be calculated from grouping attributes on V_i. The node labeled none corresponds to an empty set of group-by attributes (tuples are not grouped). The benefit of this representation is that a query can be used to answer or update another query. An extension of the data cube lattice in order to adapt it to a distributed case was proposed in [3, 53]. Indeed, the cube has been modified by adding edges that mark the derivation relationship between views on different computer nodes.

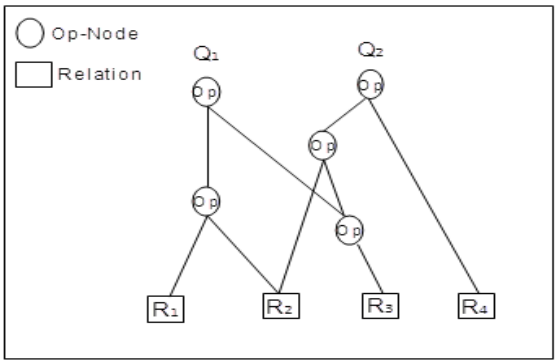


Figure 2: The MVPP of the two queries Q_1 and Q_2.

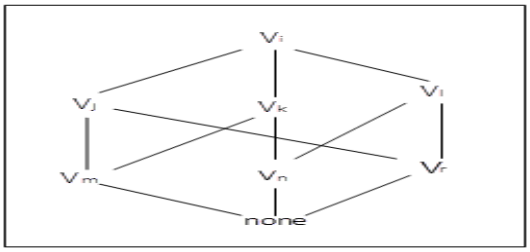


Figure 3: Sample Data Cube Lattice for eight views.

3.1.2 Query Rewriting

Query rewriting based approaches not only compute the set of materialized views but also find a complete rewriting of the queries over it. Here, the input to the view selection is not a multiquery DAG but the query definitions. The view selection problem is modeled as a state search problem using a set of transformation rules. These rules detect and exploit common subexpressions between the queries of the workload and guarantee that all the queries can be answered using exclusively the materialized views. Query rewriting based approaches not only compute the set of materialized views but also find a complete rewriting of the queries over it. Nevertheless, the completeness of the transformation rules may make the complexity of state space search strategies exponential.

3.1.3 Syntactical Analysis of the Workload

Some view selection methods are based on syntactical analysis of the workload to identify candidate views. These approaches analyze the workload and pick a subset of relations from which to materialize one or more views, if only if has the potential to reduce the cost of the workload significantly. How-

ever, the search space for computing the optimal set of views to be materialized may be very large.

3.2 Resource Constraints

Resource constraints considered during the view selection can be taken into account when classifying view selection methods. There are three main models presented in literature: unbounded, space constrained and maintenance cost constrained.

3.2.1 Unbounded

In the unbounded setting, there is no limit on available resources (storage, computation etc.). Thus, the view selection problem consists in choosing a set of views to materialize that minimizes the query processing cost and the view maintenance cost. Formally thus, the problem is:

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M) + \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)\right)$$

However, this approach may lead to two kinds of problems. First, sometimes the selected views may be too large to fit in the available space. Second, the cost of the view maintenance may offset the performance advantages provided by the view materialization.

3.2.2 Space Constrained

Due to the storage space limitation, materializing all views is not always possible. In this setting, a useful notion is that of a view benefit (or query benefit). This is defined as the reduction in the workload evaluation cost, that can be achieved by materializing this view. Also relevant in this context is the *per-unit benefit*, obtained by dividing the view benefit by its space occupancy. It has been shown [19] that the per-space unit benefit of a view can only decrease as more views are selected (monotonic property). The space constrained model minimizes the query processing cost plus the view maintenance cost under a space constraint.

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M) + \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M)\right)$$

$$\text{under } \sum_{V_i \in M} size(V_i) \leq S$$

where S is the storage space capacity.

3.2.3 Maintenance Cost Constrained

This model constrains the time that can be allotted to keep up to date the materialized views in response to updates on base relations. In the maintenance cost constrained model, the maintenance cost of a view may decrease with selection of other views for materialization. Therefore, the query benefit per unit of maintenance cost of a view can increase [20]. This non monotonic nature of maintenance cost makes the view selection problem more difficult. The maintenance cost constrained model minimizes the query processing cost under a maintenance cost constraint.

$$minimize \left(\sum_{Q_i \in Q} f_{Q_i} * Qc(Q_i, M)\right)$$

$$\text{under } \sum_{V_i \in M} f_u(V_i) * Mc(V_i, M) \leq U$$

where U is the view maintenance cost limit.

The models that we have presented in section 3.2 can be extended to the distributed setting by taking into account the distributed specific features (i.e., the communication cost between the computer nodes).

4. REVIEW OF VIEW SELECTION METHODS

In this section, we classify the view selection methods according to several dimensions characterizing their algorithms (i) resource constraints they consider during the view selection process and (ii) frameworks they use to obtain the candidate views (see figure 4). Based on this classification, we review most of the view selection methods. The best-known heuristic algorithms proposed in litterature to solve the view selection problem, namely: deterministic algorithms, randomized algorithms, hybrid algorithms or constraint programming.

4.1 Deterministic Algorithms Based Methods

Much research work on view selection uses deterministic strategies to address the view selection problem. [41] is the first paper that provides a solution for materializing view indexes which can be seen as a special case of the materialized views. The solution is based on A* algorithm [37]. An exhaustive approach is also presented in [31, 39] for finding the best set of views to materialize. Nevertheless, an exhaustive search cannot compute the optimal solution in a reasonable time.

The authors in [22] present and analyze algorithms for view selection in case of OLAP-style queries. They provide a polynomial-time greedy algorithm to select a set of views to materialize that minimizes the query processing cost subject to a space constraint. However, this approach does not con-

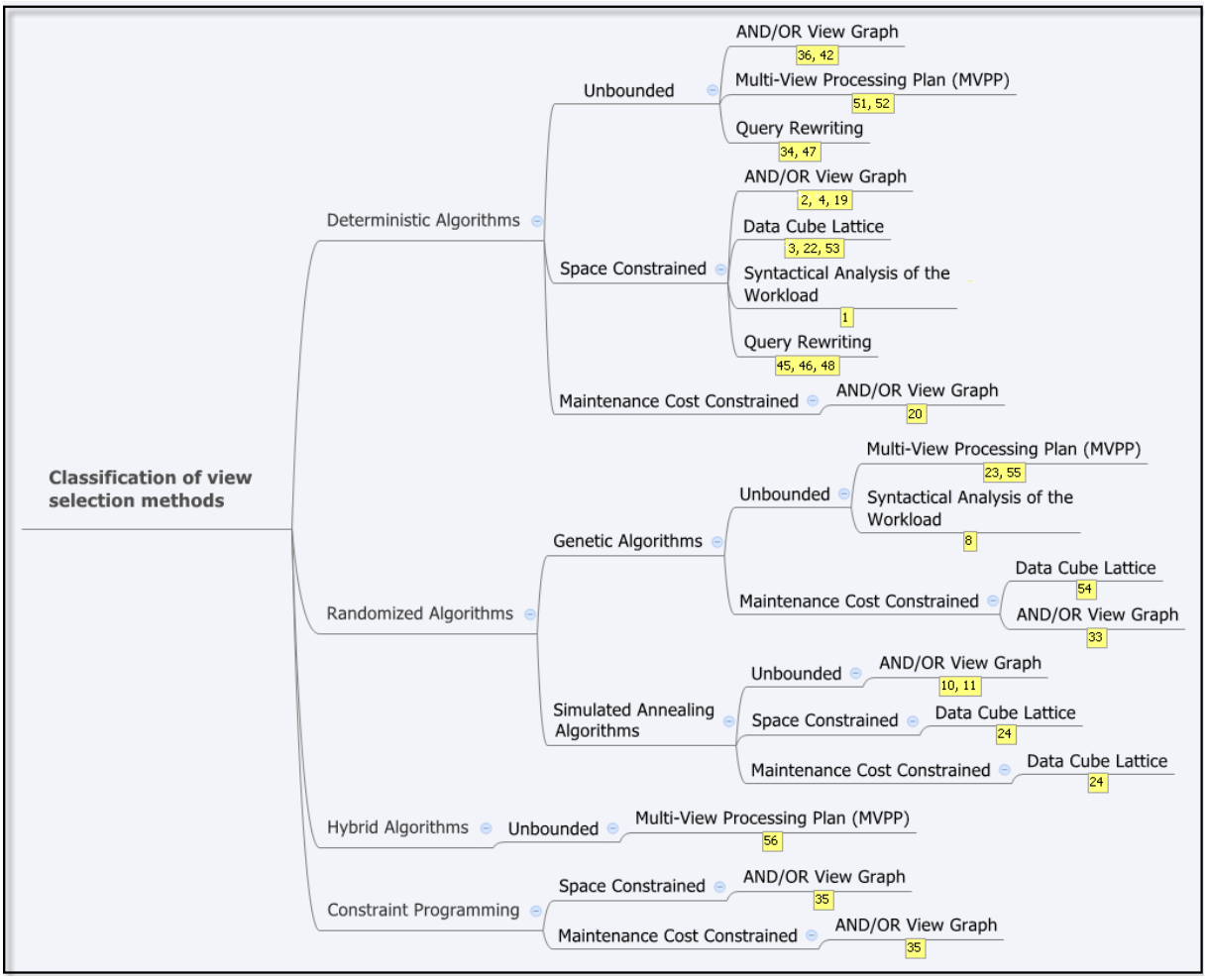


Figure 4: A Classification of view selection methods.

sider the view maintenance cost. The work in [51] is dealing with more general SQL queries which include select, project, join, and aggregation operations. A greedy algorithm has been designed to select a set of materialized views so that the combined query processing and view maintenance cost is minimized. However, the view maintenance cost has been overrated since the maintenance cost for a materialized view is the cost used for constructing this view. Besides, the view selection is done without any resource constraint.

A theoretical framework for the view selection problem in data warehousing setting has been developed in [19]. Their work provides a near-optimal exponential time greedy algorithm for the case of AND-OR view graph and near-optimal polynomial time greedy algorithm for the cases of AND view graph and OR view graph. This approach was extended in [20] to study the view selection under a maintenance cost constraint.

The authors in [42] demonstrate that using multi-query optimization techniques in conjunction with a greedy heuristic is practical and provides significant benefit. The greedy heuristic is used to iteratively pick from the AND-OR view graph the set of views to materialize that minimizes the query processing cost. This study was extended in [36] to consider how to optimize view maintenance cost. In addition to speed up the query workload by selecting materialized views, algorithms exploit common sub-expressions between view maintenance expressions to compute an efficient plan to the maintenance of the materialized views. However, the view selection has been studied without any resource constraint.

The view selection algorithm proposed in [2] is based on the notion of level in the query tree (each view of the query tree is associated to a level). In this approach, the view selection problem is studied under a space constraint and solved in two phases. The first phase depends on local optimization by

taking each query and pre-selecting a set of views which reduce the query processing cost without increasing significantly the view maintenance cost. The second phase computes the cost for each level of the query graph and selects the one which has the minimal sum of query processing and view maintenance cost.

The view selection has been studied in [34, 45, 46, 47, 48] under the condition that the input queries can be answered using exclusively the materialized views. An exhaustive algorithm has been designed in [47] to select a set of materialized views while minimizing the combination of the query processing and view maintenance cost. This work was extended in [34] by developing greedy algorithms that expand only a small fraction of the states produced by the exhaustive algorithm. The view selection problem in [45, 46, 48] is addressed under a space constraint. However, their view selection algorithm is still in exponential time. A survey of work on answering queries using views can be found in [21].

The study in [1] is based on a syntactical analysis of the workload to address the problem of selecting both views and indexes to be materialized. This approach proceeds in three main steps. The first step analyses the workload and chooses subsets of base relations with a high impact on the query processing cost. Based on the base relations subsets, the second step identify syntactically relevant views and indexes that can potentially be materialized. In the third step, the system runs a greedy enumeration algorithm to pick a set of views and indexes to materialize based on the result of the second step by taking into account the space constraint. Nevertheless, this approach does not take into account the view maintenance cost.

The works published in [3, 53] address the view selection problem in a distributed data warehouse environment. An extension of the concept of a data cube lattice to capture the distributed semantics has been proposed. Moreover, they extend a greedy based selection algorithm for the distributed case. However, the cost model that they have used does not include the view maintenance cost. Furthermore, the network transmission costs are not considered which is very important in a distributed context. Indeed, the communication cost is computed only from the size of the query result.

The above methods take a deterministic approach either by exhaustive search or by some heuristics such as greedy. However, greedy search is subjected to the known caveats, i.e., sub-optimal solutions may be retained instead of the globally optimal one since initial solutions influence the solution greatly. As a result, many paradigms and programming techniques have been developed to improve the solutions of the view selection problem: randomized algorithms, hybrid algorithms and constraint programming which we describe in next subsection.

4.2 Randomized Algorithms Based Methods

Typical randomized algorithms are genetic [14] or use simulated annealing [30]. Genetic algorithms generate solutions using techniques inspired by the natural evolution process such as selection, mutation, and crossover. The search strategy for these algorithms is very similar to biological evolution. Genetic algorithms start with a random initial population and generate new populations by random crossover and mutation. The fittest individual found is the solution. The algorithms terminate as soon as there is no further improvement over a period.

A genetic algorithm has been used in [23, 55] in conjunction with the MVPP framework to solve the view selection problem. The materialized views have been selected according to their reduction in the combined query processing and view maintenance cost. However, because of the random characteristic of the genetic algorithm, some solutions can be infeasible. For example, in the maintenance cost constrained model, when a view is selected, the benefit will not only depend on the view itself but also on other views that are selected. One solution to this problem is to add a penalty value as part of the fitness function to ensure that infeasible solutions will be discarded. For instance, a penalty function has been applied in [33] which reduces the fitness each time the maintenance cost constraint is not satisfied. This approach minimizes the query processing cost given varying upper bounds on the view maintenance cost, assuming unlimited amount of storage space. In order to let the genetic algorithm converge faster, they represent the initial population as a favorable configuration based on external knowledge about the problem and its solution rather than a random sampling, i.e., the views with a high query frequency are most likely selected for materialization. However, the genetic algorithm may tend to get stuck at a poor local optimum fairly early. A solution was provided in [54] to avoid premature convergence and keep improving the solution by incorporating constraints into the algorithm through a stochastic ranking procedure where no penalty functions are used.

The study presented in [8] which is based on a syntactical analysis of the workload deals with the

distributed view selection problem. This approach consists of three main steps. The first one extends the base relations selection algorithm described in [1] for the distributed scenario. Based on the result of the first step and the similarity between queries, the second step generates the candidate views which are promising for materialization. In the third step a genetic algorithm is applied to select a set of materialized views and the nodes of the network on which they will be materialized that minimize the query processing and view maintenance cost. However, this approach does not take into account either the space constraint or the maintenance cost constraint.

The approaches proposed in [10, 11, 24] use simulated annealing algorithms to address the view selection problem. These algorithms are motivated by an analogy to annealing in solids. Simulated Annealing algorithms start with an initial configuration, generate new configurations by random walk along the different solutions of the solution space according to a cooling schedule and terminate as soon as no applicable ones exist or lose all the energy in the system.

Materialized views have been selected in [10] so that the combined query processing and view maintenance cost is minimized. The view selection problem is solved in [24] under the case where either the space constraint or the maintenance cost constraint is considered. Further, randomized search has been applied to solve two more issues. First, they considered the case where both space and maintenance constraints exist. Next they applied randomized search in the context of dynamic view selection.

In order to support the scalability when the number of views and queries become large, a new approach has been introduced in [11] using Parallel Simulated Annealing (PSA) for materialized view selection. By performing simulated annealing with multiple inputs over multiple compute nodes concurrently, PSA is able to improve the quality of obtained sets of materialized views. Moreover, PSA is able to perform view selection on MVPP having a much larger number of views, which reflects the real data warehousing environment. However, the view selection problem is solved without any bound neither on the storage space nor on the view maintenance cost.

Randomized algorithms can be applied to complex problems dealing with large or even unlimited search spaces. Thus, the use of randomized algorithms can be considered in solving large combinatorial problems such as the view selection problem. However, the quality of the solution [1] depends on the set-up of the algorithm as well as the extremely difficult fine-tuning of algorithm that must be performed during many test runs.

4.3 Hybrid Algorithms Based Methods

Hybrid algorithms combine the strategies of deterministic and randomized algorithms in their search in order to provide better performance in terms of solution quality. Solutions obtained by deterministic algorithms are used as initial configuration for simulated annealing algorithms or as initial population for genetic algorithms.

A hybrid approach has been applied in [56] which combines heuristic algorithms i.e., greedy algorithms and genetic algorithms to solve three related problems. The first one is to optimize queries. The second one is to choose the best global processing plan from multiple processing plans for each query. The third problem is to select materialized views from a given global processing plan. Their experimental results confirmed that hybrid algorithms provide better performance than either genetic algorithms or heuristic algorithms i.e., greedy algorithms used alone in terms of solution quality. However, their algorithms are more time consuming and may be impractical due to their excessive computation time.

4.4 Constraint Programming Based Methods

Constraint programming is a descendant of declarative programming. This programming technique has been exploited in many applications for solving combinatorial problems [49]. The success of using constraint programming for combinatorial optimization is due to its combination of high level modeling, constraint propagation and facilities to control the search behavior.

A constraint programming based approach has been presented in [35] to address the view selection problem. More specifically, the view selection problem has been modeled as a constraint satisfaction problem. Its resolution has been supported automatically by constraint solver embedded in the constraint programming language. The authors proved experimentally that a constraint programming based approach provides better performances compared with a randomized method i.e., genetic algorithm in term of cost savings. The view selection has been studied under the case where (i) only the maintenance cost constraint is considered and (ii) both

[1]The solution quality represents the quality of the set of materialized views found by the algorithm. For example, the solution quality may be measured in term of cost savings.

maintenance cost and space constraints exist. They have also shown that their approach is scalable.

5. CONCLUSION

This study provides a critical survey of different approaches in which the view selection has been studied in relational databases and data warehouses as well as in a distributed setting. We have defined formally the view selection problem and identified the main view selection dimensions along with view selection methods have been classified. Based on the classification, we have discussed most of view selection methods.

Analysis of state of the art of view selection has shown that there is very few work on view selection in distributed databases and data warehouses [3, 8, 53] and no effective solution for peer to peer systems. Indeed, [16] seems to be the only paper which deals with the view selection problem in peer to peer environment. In fact, it is provided a full definition of the problem but without providing any algorithm or detail on how to select an effective set of views to materialize and place them at appropriate peers. Thus, one of challenging directions of future work aims at addressing the view selection problem in a distributed setting. More recently, materialized view selection has been explored in semantic web databases [7, 15] in order to facilitate efficient processing of RDF queries and updates. However, they consider a static workload which contradicts the dynamic nature of the web. Indeed, any change to the workload should be reflected to the view selection as well. This issue will be the future aspect while studying the view selection in semantic web databases.

6. ACKNOWLEDGEMENTS

We would like to thank the reviewers for their valuable comments to improve this paper.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and V.R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, pages 496–505, 2000.

[2] X. Baril and Z. Bellahsene. Selection of materialized views: A cost-based approach. In *CAiSE*, pages 665–680, 2003.

[3] A. Bauer and W. Lehner. On solving the view selection problem in distributed data warehouse architectures. In *SSDBM*, pages 43–, 2003.

[4] Z. Bellahsene, R. Coenmans, and J. Tranier. Matérialisation de vues dans les entrepôts de données. une approche dynamique. *Ingénierie des Systèmes d'Information*, 11(6):33–53, 2006.

[5] Z. Bellahsene, M. Cart, and N. Kadi. A cooperative approach to view selection and placement in P2P systems. In *OTM*, pages 515–522, 2010.

[6] R.G. Bello, K. Dias, A. Downing, J. Feenan, J.L. Finnerty, W.D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in ORACLE. In *VLDB*, pages 659–664, 1998.

[7] R. Castillo, and U. Leser. Selecting materialized views for RDF data. In *ICWE Workshops*, 2010.

[8] L.W.F. Chaves, E. Buchmann, F. Hueske, and K. Böhm. Towards materialized view selection for distributed databases. In *EDBT*, pages 1088–1099, New York, NY, USA, 2009. ACM.

[9] R. Chirkova, A.Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *VLDB J.*, 11(3):216–237, 2002.

[10] R. Derakhshan, F.K.H.A. Dehne, O. Korn, and B. Stantic. Simulated annealing for materialized view selection in data warehousing environment. In *Databases and Applications*, pages 89–94, 2006.

[11] R. Derakhshan, B. Stantic, O. Korn, and F.K.H.A .Dehne. Parallel simulated annealing for materialized view selection in data warehousing environments. In *ICA3PP*, pages 121–132, 2008.

[12] P. Deshpande, K. Ramasamy, A. Shukla, and J.F. Naughton. Caching multidimensional queries using chunks. In *SIGMOD Conference*, pages 259–270, 1998.

[13] C.A. Dhote, and M.S. Ali. Materialized View Selection in Data Warehousing: A Survey. In *Journal of Applied Sciences*, pages 401–414, 2009.

[14] D.E. Goldberg. Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley, 1989.

[15] F. Goasdoue, K. Karanasos, J. Leblay, and I. Manolescu. View Selection in Semantic Web Databases. In *PVLDB*, pages 97–108, 2011.

[16] S.D. Gribble, A.Y. Halevy, Z.G. Ives, M. Rodrig, and D. Suciu. What can database do for peer-to-peer? In *WebDB*, pages 31–36, 2001.

[17] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[18] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *SIGMOD Conference*, pages 157–166, 1993.

[19] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.

[20] H. Gupta and I.S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT*, pages 453–470, 1999.

[21] A.Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[22] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *SIGMOD Conference*, pages 205–216, 1996.

[23] J.T. Horng, Y.J. Chang, and B.J. Liu. Applying evolutionary algorithms to materialized view selection in a data warehouse. *Soft Comput.*, 7(8):574–581, 2003.

[24] P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *Data Knowl. Eng.*, 42(1):89–111, 2002.

[25] P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. In *SIGMOD Conference*, pages 25–36, 2002.

[26] H.J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *PODS*, pages 167–173, 1999.

[27] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

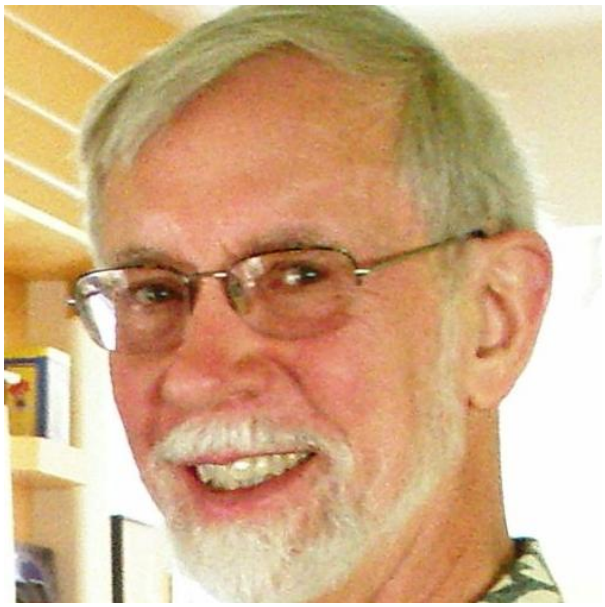
[28] D. Kossmann, M.J. Franklin, and G. Drasch. Cache investment: integrating query optimization and distributed data placement. *ACM Trans. Database Syst.*, 517–558, 2000.

[29] Y. Kotidis and N. Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In *SIGMOD Conference*, pages 371–382, 1999.
[30] P.J.M. Laarhoven and E.H.L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.
[31] W. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *ICDE*, pages 277–288, Washington, DC, USA, 1997. IEEE Computer Society.
[32] A. Labrinidis , Q. Luo , J. Xu, and W. Xue. Caching and Materialization for Web Databases. In *Foundations and Trends in Databases*, pages 169–266, 2009.
[33] M. Lee and J. Hammer. Speeding up materialized view selection in data warehouses using a randomized algorithm. *Int. J. Cooperative Inf. Syst.*, 10(3):327–353, 2001.
[34] S. Ligoudistianos, D. Theodoratos, and T.K. Sellis. Experimental evaluation of data warehouse configuration algorithms. In *DEXA Workshop*, pages 218–223, 1998.
[35] I. Mami, R. Coletta, and Z. Bellahsene. Modeling view selection as a constraint satisfaction problem. In *DEXA*, pages 396–410, 2011.
[36] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD Conference*, pages 307–318, 2001.
[37] N.J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.
[38] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *PDIS*, pages 158–169, 1996.
[39] K.A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD Conference*, pages 447–458, 1996.
[40] N. Roussopoulos. The logical access path schema of a database. *IEEE Trans. Software Eng.*, 8(6):563–573, 1982.
[41] N. Roussopoulos. View indexing in relational databases. *ACM Trans. Database Syst.*, 7(2):258–290, 1982.
[42] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Conference*, pages 249–260, 2000.
[43] P. Scheuermann, J. Shim, and R. Vingralek. Watchman : A data warehouse intelligent cache manager. In *VLDB*, pages 51–62, 1996.
[44] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *SIGMOD Conference*, pages 793–795, 2006.
[45] D. Theodoratos, S. Ligoudistianos, and T.K. Sellis. Designing the global data warehouse with SPJ views. In *CAiSE*, pages 180–194, 1999.
[46] D. Theodoratos, S. Ligoudistianos, and T.K. Sellis. View selection for designing the global data warehouse. *Data Knowl. Eng.*, 39(3):219–240, 2001.
[47] D. Theodoratos and T.K. Sellis. Data warehouse configuration. In *VLDB*, pages 126–135, 1997.
[48] D. Theodoratos and T.K. Sellis. Data warehouse schema and instance design. In *ER*, pages 363–376, 1998.
[49] M. Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996. 10.1007/BF00143881.
[50] J. Widom. Research problems in data warehousing. In *CIKM*, pages 25–30, 1995.
[51] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.
[52] J. Yang, K. Karlapalem, and Q. Li. A framework for designing materialized views in data warehousing environment. In *ICDCS*, 1997.
[53] W. Ye, N. Gu, G. Yang, and Z. Liu. Extended derivation cube based view materialization selection in distributed data warehouse. In *WAIM*, pages 245–256, 2005.
[54] J.X. Yu, X. Yao, C.H. Choi, and G. Gou. Materialized view selection as constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 33(4):458–467, 2003.
[55] C. Zhang and J. Yang. Genetic algorithm for materialized view selection in data warehouse environments. In *DaWaK*, pages 116–125, 1999.
[56] C. Zhang, X. Yao, and J. Yang. An evolutionary approach to materialized views selection in a data warehouse environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 31(3):282–294, 2001.
[57] J. Zhou, P. Larson, J. Goldstein, and L. Ding. Dynamic materialized views. In *ICDE*, pages 526–535, 2007.
[58] Y. Zhuge, H.G. Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *SIGMOD Conference*, pages 316–327, 1995.

David Lomet Speaks Out

On database recovery, logs, versions and more…

by Marianne Winslett and Vanessa Braganholo



David Lomet

http://research.microsoft.com/~/lomet/

Welcome to this installment of ACM SIGMOD Record Series of Interviews with Distinguished Members of the Database Community. I am Marianne Winslett, and today we are in Providence, site of the 2009 SIGMOD and PODS conference. I have with me today David Lomet, who has been a principal researcher and the manager of the Database Group at Microsoft Research since 1995. Before joining Microsoft, Dave worked at Digital Equipment Corporation, Wang Institute, and IBM Research in Yorktown. Dave's research interests lie in access methods, concurrency control, and recovery. He has received two SIGMOD Best Paper awards, and he is an ACM Fellow and an IEEE Fellow. Dave's PhD is from the University of Pennsylvania. So Dave, welcome!

Thanks, it's nice to be here.

Dave, I read that you are one of the people who invented transactions. I have to ask – how many people invented transactions?

Well, it is really very hard to count. The first people who really used something similar to a transaction were the IMS folks -- Ron Obermark, and the IMS program isolation feature. But even before that, there was old master and new master tapes, where in fact the rollback was the old master and the committed transaction was the new master. So, there is a lot of people who played a roll. I went to Newcastle [University in Britain working on Brian Randell's Reliability Project] early in my career, and they had something called recovery blocks, which was a

mechanism, but it actually implemented transactions. So I sort of provided the abstraction there for that. Jim Gray and his colleagues in San Jose provided the abstraction for the IMS program isolation facility. So, there are many inventors of transactions, but I was a contributor.

You have over 40 patents. So you must write patents at the rate that many of us write papers. Can you compare the effort involved?

It is much more tedious, actually. It is not as much of an effort as writing a paper, because, what I frequently do is I write the paper, attach the front piece patent form that goes with it, and hand that in. And that is sort of the easy part. That initiates the process. The more tedious part is having to interact with the lawyers, sometimes over an extended period of time. Some of the lawyers are marvelously good. In fact, one of them was so good that after he wrote up a description of the invention, he didn't discover a bug, but I discovered a bug having looked at his write-up. But other patent attorneys do what might be called syntactic transformation and don't always get it quite right, so it is kind of tedious sometimes. It is a mixed bag, but I think the patenting process is important if you are an industrial researcher because it is something which the company gets out of your efforts in some sort of a direct fashion, and so I believe that's a really important part of industrial research.

I think a lot of universities now would like the professors to be patenting at the same rate that you guys are.

It's expensive. It's tens of thousands of dollars. It's not a few thousands of dollars. It's tens of thousands of dollars, so it is expensive, so even Microsoft has a budget. We get budgeted for patents. So it is expensive.

Let's talk about wrong directions and missed opportunities. Where have we gone wrong in the past?

Well, I have a long career, dating back to a pre-database era in my career. One effort that I was involved with at IBM research was on a taskforce to take a look at an area called provably secure operating systems. It was an initiative by the government to make managing their multi-level secure systems somewhat easier by permitting people to run jobs at multiple security levels simultaneously on a time sharing system. But they were very concerned about the security aspects, and information leaking from the more secure side to the less secure applications. So they had a program to do provably secure operating systems, where they wanted to be able to prove that the operating system was secure. That was much simpler than trying to prove that it was correct in all its aspects, and security requirements were very well defined. But even so… So we went around, and we looked at a bunch of people who were doing program proving technology, a lot of interesting stuff. But we concluded that the prospect of any near term success in proving something as a large operating system to be secure was decidedly misguided. So we wrote a report. The end result of that report was that work in the area pretty much ceased. So that was one of my bigger negative accomplishments -- to redirect people away from that area.

All that stuff in the MOS databases must have come after that, because that was in the '80s. A lot of money went down that drain. Did people listen to your report then?

At the time, which was in the early '80s, yes. So people don't do provable secure operating systems now, surprise, I know. There is some work in multilevel secure database systems.

Not anymore, though.

Yeah. That's right, but there was some. It is a hard problem. We will probably revisit it again, and maybe at some point the state of the art of program proving will be sufficient to really do the job. My guess is that we're still not there yet.

So that would fall under the category of wrong directions rather than missed opportunities. Do you have missed opportunities also?

I've always been a big fan of something akin to capability machine hardware. It was explored in the '70s. I've also been a fan of strongly typed, low-level programming languages, in which you might be able to actually write an operating system or to write a database system in, but nonetheless, you had the type checking safety of the full mechanisms where there was no escape. We haven't pursued that. We haven't pursued that with any real vigor, and the end result is that I think we pay for it in systems which have mistakes that could have been caught had we just used the tools that we already know how to do. But we continue with these less optimal solutions, in part because we have a lot of historical code, and nobody really has the money or the time to go back and do it again, but that is a missed opportunity.

So are you a fan of these more secure versions of operating systems (like SELinux)?

I don't have very much exposure to them. I am sort of in the Microsoft community, and we have some efforts within Microsoft research exploring some of these directions, but they have yet to have real impact on the products.

You worked on product development, in architecture, programming languages, and distributed systems. Looking back, do you consider that time well spent?

I thought that all the times that I spent actually was pretty well spent. Even though actually not too much came from it, in terms of products (interesting commentary isn't it?). But, I learned a lot. I learned a lot from the developers. I met the developers, and that is really an important thing to do. Especially if you are in industrial research, because the more ties you have with the developers, the more likely you are to be able to influence what they do, and get your research into product, which is part of what we are supposed to be doing. And so I've always thought that… In fact, let me combine this with the answer to a later question which you might ask, which is my advice to younger researchers, which is: spend some time in industrial organizations, even as a visiting researcher or whatever. Spend some time with developers learning what they do, seeing what they do, understanding how they make decisions, and understanding the problems that they're facing and you'll be better for it. You'll be able to take that more into account in your research, and your research will be more well-grounded on things which might actually have impact.

For performance reasons, traditionally the DBMS code for recovery, concurrency control and access methods is tightly bound together. In your Deuteronomy project, you have been working on unbundling that functionality and pulling it apart into pieces. How can you do that now, when people couldn't before?

So it's not just for performance reasons that we do it. We really did not understand how to do it. We do it because the locking and the logging are both done when the code is in the page looking at the records. So you know exactly what page you are on, you know exactly what record you are dealing with, you can look at surrounding records, and you already have the page latched, so you know that what you do in that page is already protected by a lock. We didn't know how to split those things apart.

Don Batory had a project years ago called GENESIS in which he explored putting various architectural pieces of database systems into sort of an erector set of database pieces, which he put together. But he never succeeded in separating the concurrency control and recovery from the data management of this. "I had this blinding insight", he says, "that if you looked at a database kernel as a distributed system, and thought about as a maintaining the state on the log, and in the database, and that that state had to be reconciled, then you could apply the kind of architectural principles, recovery principles, that we had used in a previous work that we had done on distributed recovery". We could use that to in fact solve the problem of having to be able to pull apart the database kernel. It is still an open question whether or not we'll get performance which is good enough, which is one of the reasons why we have projects going on, and we'll have another one this summer on exploring this area, but we couldn't even do it before. We really didn't have the underlying technology and understanding to be able to do it before, and now we can, and we are going to see. We're going to see how it works out.

Let's talk about time travel databases: databases that keep all past versions of their tuples. Why would we want to do that, (I think many of our readers won't know why we would want to keep all that stuff) and can we afford to do that?

So let me do the "can we afford to do it" first, because that is really easy. We've got this capacity running out of our ears, right. We have disks sitting idle with unused capacity, and this is especially true for all OLTP systems, where people buy additional disks to get the access arms. So there is just this enormous storage which is going to waste. The wonderful thing about keeping history is that you can keep it, and it is relatively cold data, it is not accessed all that often. So it doesn't really interfere that much with the access patterns for the current data. So, keeping it is an easy thing, it's a cheap thing. It's frequently just unused capacity. So that's one thing. So why might we do it? Well people like to do run ups of business trends, people like to do analysis of stock trends, people like to do audit compliance (a very big aspect of this), so they have to keep historical data. I think my research has shown that you can actually do the transaction time database stuff with minimal impact on the performance of the current system. So, it's just a natural and it's a useful function that people will have, and they'll do even more with it once they have it. I'm hopeful, though nothing has been carved in stone yet.

In the olden days, how did the detailed data get from the production database to data warehouse?

People would run scripts, which would funnel the data (frequently from multiple sources), cleaning it up, bringing it into the warehouse, as a separate, sort of offline process, for the most part.

But they were lacking the level of detail that they will have now, with all the past versions?

You could always do what I've just described by keeping the log. The log has all the information on it. The question is one of convenience, and I remember Bruce Lindsay one time saying that the database is the log -- what we keep in terms of state is simply an optimization. So you can think about the transaction time stuff in exactly the same way -- it is an optimization. You can use it to do online backups, and have the additional facility of having it be a queryable backup. So there is a lot of things you can use it for. You can use it to recover from bad user transactions, and do that without the enormously expensive operation of point-in-time recovery, which requires you to install a backup, grow it forward, de-commit scores of transactions. And you can do all that, and having a transaction time database makes that easy. So there's a lot of reasons: functional reasons, infrastructure reasons, and it's cheap. The capacity is there anyway, and the cost is modest.

Ok, so, how do you keep all those old cold historical tuples from clogging up your hot pages, and messing up your performance on your hot pages?

You use the time-split B-tree, and you do page splits, and move them to historical portions of the disk. That works really easily. In fact, Mike Carey had a paper[1], probably in the early '90s, I think, on using multi-versions, and he described the number of properties of how you migrate versions out of the current page. His was a more generic statement of it, but the time-split B-tree does exactly what he did, move it in bulk. You move it periodically, and you keep the versions close for a while, but you age them out of the current database. And with the time-split B-tree you can index them as well. So it works pretty well.

Many companies desperately need dependable systems – for example, Amazon, with their website for purchases, Travelocity, and Wall Street firms. So why haven't we computer scientists been able to give them dependable systems?

Well, they have reasonably dependable systems. Most of what they have sort of grew organically. The way a lot of development works is people build a system, they try to get the functionality right, it starts operating… They then realize that they've got some sort of dependability problem, and so they start adding it. It sort of grows much like the genetic code. You know, systems fail, and they fix them, and they get more reliable over time.

Computer scientists have the luxury of stepping back and looking at the process and trying to understand what general principles work and what can guide that sort of activity. They have been doing that for a while. I mean, there were TP systems years ago that had a strategy called stateless applications, and you ran everything inside of a transaction. The line of work that we were pursuing doesn't require that you run everything inside of a transaction. In fact, it solves

[1] Paul M. Bober, Michael J. Carey: Multiversion Query Locking. VLDB 1992: 497-510.

some of the problems that running things inside of a transaction have trouble dealing with, such as, what do you do when a transaction refuses to commit? In our systems, because we keep applications persistent without using transactions, there is a place where you can write a programmable piece of code which will deal with those kinds of errors, whereas in the TP systems there tends to be no place where you can be sure that it's going to be stably available, because stability is guaranteed by the transaction mechanism itself. So we think that the Phoenix project, which is the earlier project that I did at Microsoft, is a way of dealing with some of these issues. And it's also a way of making the process a little less expensive, because, transactions are a rather expensive mechanism when what you really want is just to keep the application running across crashes.

So how do you allow the application to survive system crashes if you don't have transactions?

Well, it's like the redo part of transaction recovery. You log the nondeterministic actions that interrupt the flow of the application, and you make some assumptions, which of course you have to enforce, that applications are what are called piece-wise deterministic, which is to say they'll execute deterministically between these interactions that you've captured. Then, if the application fails, you can use the log, which has captured the nondeterministic activities, and replay those instead of the actual activities at the appropriate points when the application calls for them. And so, you can recreate the application state simply by the replay of the logged events against the re-executed application.

Is that transparent to the application developer in the way that the transaction almost are?

You can make it transparent. There are a lot of efforts that people have made, some of which are not entirely transparent, but what we did in the Phoenix project was, we intercepted messages between the distributed components of the systems. We logged those messages -- that was the source of the nondeterminism. That interception mechanism was all captured inside of the .NET remoting runtime system, and so all the user saw was the normal .NET remoting, and all our logging was done sort of transparently. So this was truly transparent to the application. The only thing they need to do was to declare that their application was to be persistent, and then put it in, if you will, the persistent directory, and then we made sure that it was.

Has that made it into any products?

It's a sad story; a sad story of almost, but the answer is no. We came very close. In fact, we demoed the system at one of Microsoft's TechFests, and a guy from the Gardner group was there and predicted that with 70% probability it would appear in a product by 2007. We didn't quite make that. But it was close.

Are database researchers paying enough attention to multicore architectures?

Oh, I think the answer is yes, they are paying plenty of attention to it. It is hard to get leverage on it, in fact it is a very hard problem, and we'll be working on it for a long time.

What do you think about cloud computing and databases?

So unlike the situation with multicore, where everybody has access to multicore machines, not everybody has access to a cloud infrastructure. So, it is much harder to do research on the cloud. In fact, in our research group, we really don't do research specifically oriented towards the cloud. My participation in cloud computing involved an effort to help a product group do something on a cloud database system, and that's become an offering for SQL data services. But it is very hard for people outside of industrial organizations to have access to the kind of infrastructure that you need to really do research on cloud databases. You can sort of play with the Amazon type of infrastructure, and things of that sort, but it is really hard to do a fully robust database system of the sort that you might want to have taken seriously as a database system. So cloud computing is much harder, cloud databases is much harder to do research on. So we probably aren't doing enough, but there are technical difficulties in being able to do it, which I don't really have any advice or suggestions as to how to deal with it.

Well maybe we should say, so what do you think are the main research issues for database computing in a cloud?

I think we've got to change the way we think about making transactions stable. I think we have to move to a replication model, not simply a commit the log and put it on disk, because the availability requirements are such that you need instant fall over.

Does that mean replicated logs, or do you mean replicating the things that are actually doing the work?

Replicating the database, so that in fact the only outage you have is the amount of time that it takes to switch you from one replica to the other.

Do we already know how to do that from Tandem?

We know a little bit about it, but that is not to say we know as much as we should. We know a little bit about quorum protocols, and things of that sort. But we're going to be living in a world where one of the features of the cloud is the intense desire to try to control the costs of the infrastructure; which means that people are going to be buying what we refer to as el-cheapo machines. El-cheapo machines have el-cheapo disks on them, and el-cheapo disks fail at somewhat higher rates than the sort of the higher powered, more expensive disks do. We need to be able to have the strategies for coping with those, and making failure operations be a normal event as opposed to an extraordinary event.

So do we also need to replicate at higher levels, or you are only worried about the storage itself? What about the servers themselves?

I anticipate that in a system where people are really concerned about having a high availability all the time, that we'll replicate the applications.

What role do you see flash memory playing in the database world over the next ten or so years?

Well, people are certainly working on it, and again, this does not pose the same problems that cloud computing does. I mean, people have easy access to flash. What they don't have access to is raw flash, but they certainly have access to flash disks. The trick will be to see how we can successfully exploit it. I've seen a number of papers in which we've started to make some inroads on the problem, but it seems pretty clear to me that there is a lot more to be done. With a little bit bolder re-architecting, I think we'll exploit it more effectively. Right now we are sort of nibbling at the edges, I think.

Well, I've heard people argue that you can't just insert into your memory hierarchy memory flash disk, that that's not going to work, because of the cost performance, and that it will always take way longer to write to flash than to read, so this asymmetry makes it a little awkward for many uses. So do you think there is a right way to use flash?

I think the simplest way, one of the ways you will see people using it initially is just as a replacement for disks, for high performance disks. Flash is much more expensive in terms of its cost per byte, but in terms of its cost per IOPS, it's in fact quite competitive, and in fact does a lot better than disks. You can in fact buy IOPS more cheaply with flash than you can with disks. So those people who were buying large disks and more disks than they needed for the access time may be tempted to use flash instead.

That makes good sense. So that, in the memory hierarchy, that's replacing the bottom rung, well, depending on what you consider to be the bottom rung, it is replacing one of the rungs by flash. Is there any obvious way to add it as an additional rung somewhere, or is it just too tricky?

One of the things that I think is a bit appealing is to implement (bringing it back to my own work) a time-split B-tree where you keep the current database in flash, but when you do a time-split, you move the time-split page, the history page, to disk. That sort of gives you plenty of capacity for the keeping of the history, but it gives you very high performance and high IOPS for the current data.

And still you are not treating it as a cache for the disk, it's like a special disk.

Yes, but I think there's a bunch of stable main memory technologies which are coming along, which might in fact have the effect of replacing main memory -- currently volatile with stable main memory. That poses another interesting set of problems. This is one of these things where the hardware base seems to go along forever with the same technology, and then all of a sudden you find that there's a phase transition and everybody's using something else. It will take a little bit of getting used to. But there is an enormous commercial pressure to be out there with a database system which exploits the new technology to get a commercial edge. This is bound to be something of great interest and researchers should be taking a look at it.

You have been involved with ACM and SIGMOD and VLDB, but even more involved with the IEEE. In particular, you have been the editor-in-chief of the Data Engineering Bulletin since 1992, you are an IEEE Golden Core Member, and you received IEEE Outstanding Contribution and Meritorious Service Awards. So I think you are a good person to ask about IEEE, which I don't think we've talked about in previous interviews.

How is the Data Engineering Bulletin different from the SIGMOD Record?

That is one of my favorite topics in fact. Rakesh Agrawal was the one who recruited me as the editor of the Bulletin. Something that he emphasized, but I think it is really true, is that every issue that we do is a special issue. It is a special issue on ongoing research. It is not the kind of work that you see in the published papers -- it is more work in progress. So in some sense, it gives you a window into what's actually going on now, so that's relatively unique. Then another factor is that I try to give it an industrial slant. I try to involve people who don't usually write papers, and let the rest of the world know what they are up to as well. It's a combination of those things, plus, the set of editors I have more or less cover the field, or at least interesting areas that are now of real interest. We try very hard to keep it current, keep it with some industrial focus, and it's a great place if you are looking or want to start looking in an area and do some research in an area -- it's a great place to start to look.

How is the IEEE Technical Committee on Data Engineering different from ACM SIGMOD?

Well, that's a long, and in some sense, a sad story. SIGMOD has a budget. They have money that they roll over from one year into the next. Technical Committee within the IEEE, at the end of the year, any surplus that they have reverts to the IEEE. So the end result is that we have much less discretion in terms of what we can do, much less discretion in terms of how we run our operations, and that's by design. I'm not a big fan of that, but that's the way it is. I happen to like the ACM model better, but when you end up being involved in an organization, you sort of end up with the rules of the organization… The Technical Committee does a pretty good job, I think, in terms of sponsoring conferences and things of that sort, and they are of course the sponsor of the Data Engineering Bulletin as well, but it's more a limited role than SIGMOD would have.

If you magically had enough extra time to do one additional thing at work that you are not doing now, what would it be?

You mean aside from napping?

I think napping is a good answer!

It is a good answer! In the back of my mind I have the outlines of a paper on a full theory of recovery, which I need a co-author to do, because I am not really a theorist, but it requires somebody who's comfortable at theory. Mark Tuttle was great in that respect with the other work, but the idea here is to apply some of the work that Mark Tuttle and I had done but in the broader setting of covering application recovery type stuff as well as database type stuff.

Maybe you'll get a volunteer who reads this or sees this.

Maybe, that would be good!

If you could change one thing about yourself as a computer science researcher, what would it be?

I'll mention two things. I've seen people have success, if you will, seizing the moment, and being first in an area, and in some respects, I've recoiled from that. Maybe it's a little bit of timidity on my part, but I like to think things over, and make sure that I know what I'm doing before I rush out. It's not that these people don't necessarily know what they are doing, but usually there's a certain first cut aspect to that, which I tend to recoil from. I want a little bit more solid footing underneath myself. So that's one thing. Another thing is I probably made a mistake in getting a little too far away from programming. I don't do very much programming. It's probably a mistake, I probably should do more.

Among all your past research, do you have a favorite piece of work?

No single favorite piece, but I'll tell you what I like, and what I like is some combination of things which have an abstract aspect, a theoretical aspect, and a pragmatic engineering aspect, and hopefully some practical application. So things like some of the recovery work that I've done, the recovery theory that I did with Mark Tuttle, or the application recovery principles that I did with Roger Barga and Gerhard Weikum. Things of that sort have a real appeal to me, because they have both a practical flavor, but there is some abstract principles involved, and there's some hard engineering as well. So it sort of has it all, and I like the areas where you can bring these things together. I like the Deuteronomy project which I'm doing now exactly for the same reason. It's not always easy to bring everything together in one piece, but I like it when that happens.

We were talking earlier about your favorite plant in your yard, which you said was the redwood tree, and in many ways the redwood tree is that -- it has all the historical significance, it is esthetically beautiful, it smells good, it has it all!

In my yard, it also has a rather unique role because there are not that many redwood trees in Washington State, and I have 3 of them in my back yard.

Three! Well if they start reproducing, it could be the beginning of a whole new forest.

And my house will become a tree house!

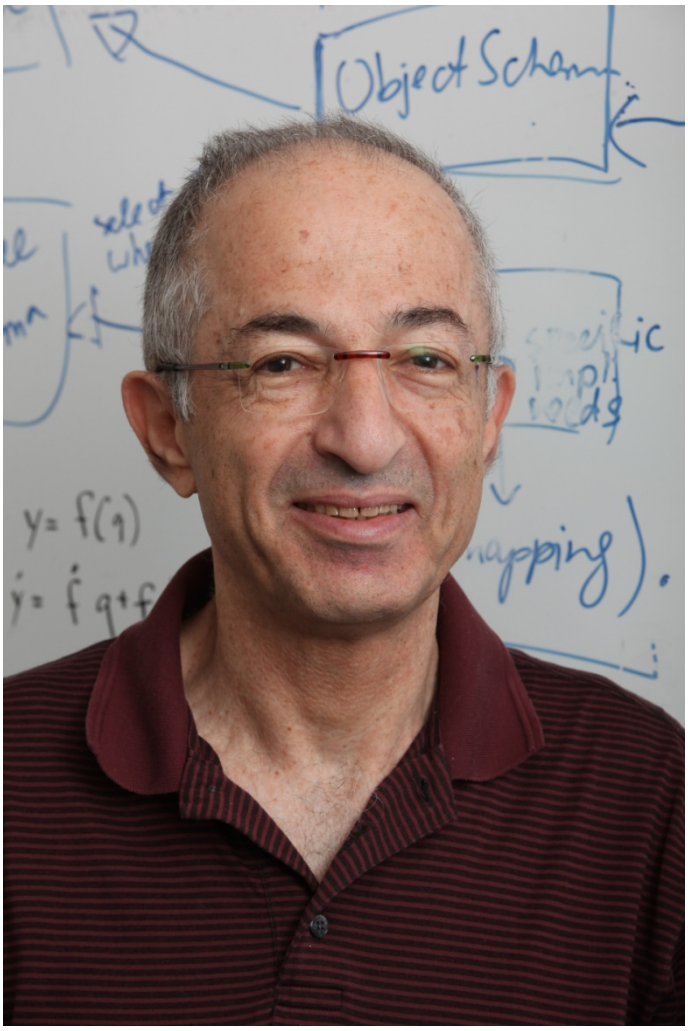
Thank you very much for talking with me today!

It's been a pleasure, thanks.

Catriel Beeri Speaks Out

On his favorite pieces of work and on the importance of Sabbaticals

by Marianne Winslett and Vanessa Braganholo



Catriel Beeri
http://www.cs.huji.ac.il/~beeri/

Welcome to ACM SIGMOD Record Series of Interviews with Distinguished Members of the Database Community. I am Marianne Winslett, and today we are in Providence, the site of the 2009 SIGMOD and PODS conference. I have with me Catriel Beeri, who is a professor of computer science at Hebrew University of Jerusalem. Catriel's research interests lie in database theory. He is a former member of the editorial board of ACM Transactions on Database Systems, and he is an ACM Fellow. Catriel's PhD is from the Hebrew University. So Catriel, welcome!

I understand that you are interested in programming languages. What do you think of our own community's programming language, SQL?

It's a nice language. I think it was nice from the beginning. Of course, now we have very thick manuals – the language has evolved in many different directions and I think that we… at least I don't find it so interesting now to read very thick manuals of SQL. But I think that the basic idea, the basic principle of SQL is the right one, and it is a very successful one. Obviously, this is the foundation for the success of database systems in the last few decades.

What do you think of this new push-for map-reduce-based programming?

I am all for it. You may know that I have been interested in functional programming languages specifically, not just in programming languages, for various reasons. One of them is because functional languages can actually express operations like map and reduce, which in some sense are more general than relational algebra, and they have very nice properties. They are completely declarative, they allow you to parallelize… So, the idea is not new... The idea of using map and reduce, or using functional languages as the basis for parallelization, it is not new, and maybe it has been waiting for the right time. And the right time is now, because we have multi-core processors, and programming languages like JAVA or C# are not going be up to the challenge of exploiting the parallelism of new hardware.

On the other hand, database people have been exploring the power of relational algebra -- not always in parallelization, of course. There have been some efforts on, you know, doing database work on many CPUs, but I think the main effort was optimization. But now, it is time to think about doing parallel optimization, I think… using parallelization for database that run on a single unit of hardware, because a single unit is going to contain many, many processors. So map-reduce and the relational algebra, I think, fit together and for me it is one paradigm. It is not that this is a new paradigm and SQL is an old paradigm. I have been looking at map-reduce at 20, 25 years ago.

I guess the map-reduce guys might argue with you and say that one of the differences SQL tries to hide from the programmer, what's efficient and what's easy to do, versus what's hard to do, like join ordering or something, and that map-reduce exposes what can be done efficiently and encourages people to program in a way that's efficient.

Maybe, but of course, once you have a nice paradigm, you can exploit the paradigm in different ways. You can try to embed the paradigm in different approaches to programming, so that they are good for different kinds of programmers or users. SQL was originally written with the intention to be used by people who don't know much about programming. It is still true that database programmers, or people who use databases, do not necessarily know much about programming. Sometimes they do, but they don't know much about query optimization, and they are not supposed to, because query optimizers are so much better than human optimizers. In this sense, I think the database story has been a success. Databases can optimize queries much better

than humans, and this obviously will be true for map-reduce. Map-reduce is parallelizable. This is one part of the new buzz word. Map-reduce runs on many, many CPUs -- it is parallelizable. And again, it is not something that a programmer is supposed to do. It is done by the machine. The machine decides how to parallelize it.

What about impedance mismatches?

As far as I know, this has not been yet solved in a satisfactory manner. Originally, people thought that object-oriented databases would do the job, and of course, lots of work went into this new kind of new systems. And then they disappeared, almost disappeared from the marketplace. Relational systems are still there. There was a lot of talk about object-relational systems, but the basis for data storage is still relational systems. From what I see on the internet, from what I hear from my students, the impedance mismatch is still there. Work is still being done. For example, the new product LINQ from Microsoft seems to be very promising because it is embedded into the programming language. It is part of the programming language, and it has some features, which are very similar to SQL. So maybe it is not the ultimate solution, but it shows that people are still trying very hard to bridge the gap, and allow seamless programming. General programming on one side, which today is object oriented, and database programming, which is relational, and somehow the effort to combine the two is still ongoing. This solution, for example, does not come from the database community. It comes from the programming language community, and most of the talk on the internet (on this issue) seems to come from people with interest in programming language design, not from the database community.

I've heard from people in industry that a lot of people use those database programming environments. They start by extracting all the data into the programming language, and then they just compute on it, so they use the database like an object store, and they don't even issue any queries at all, which I find kind of amazing.

Well, yes, but of course it depends on the expertise of the people, and the size of the system. You cannot really expect all people to use database systems in the same way. In particular, the fact that you can express so many things in SQL, so many computational processes, so that you can delegate a lot to SQL, is not something that most programmers know. Programmers in large companies, they are normally either C# or JAVA programmers. They do not know much about database programming or SQL. There use SQL to store data, simple queries, but this is it.

At SIGMOD this year, we have the Celebration of 40 Years of the Relational Model, so I think it is a good time to look back at what we've accomplished. So, among the database theory results of the past 40 years, what would you single out as having had the most impact?

Well, that is difficult to say. I think one important result, important for the database theory community, it is not clear to me whether it has been really used in the general database community, but obviously, the idea of acyclicity in queries is important. It means that queries that cannot be efficiently computed, in general, can be efficiently evaluated if they are acyclic.

So this is the general idea, and then you know, there are additions and extension, and a lot of work. Query containment, and conjunctive query evaluation are very important for many other areas of database research in general, like data exchange, data integration, evaluating queries on views, etc. So, many, many different areas actually rely on being able to evaluate conjunctive queries. So knowing about the largest sub-class, which is efficiently computable, is a nice tractable computational problem, is important.

In a completely different direction, I think that transaction management is very important. The original paper is about two phase locking. I think this is a very nice result, very interesting, very nice, very important result, even though the paper originally appeared in, I think, the Communications of the ACM, if I remember correctly.

That's ok.

Certainly, not in a database theory forum. But there has been a lot of work in the general database community and in database theory about transaction management. I think this was one of the first, maybe the first paper that showed us that you can do interesting theory. Maybe not the first one, but the one that actually attracted a lot of attention, and obviously it had a lot of practical impact. OK, so if you are talking about important results, I think that acyclicity is important, for me, data dependencies and the chase are important.

OK, tell our younger readers about data dependencies and the chase.

Well, the story actually started when I was a visitor at Princeton, in '78. I worked on multivalued dependencies at the time, and I got Jeff Ullman involved also in the story. Then, we started to look on the subject of lossless decompositions. I think the chase came up in a very small way even in the paper we wrote at that time on lossless decompositions, but then Jeff involved other students, like Alberto Mendelzon, David Maier, Sagiv and Yannakakis. The first three wrote a paper about the chase[1], and I think this was the first time the chase formally appeared in print.

Then I went back to Jerusalem and I was very lonely, nobody to talk to: nobody in databases, and very few people in computer science. I was trying to continue the work in data dependencies, and database design. I did some work on design, some materialized, some not… The work on data dependencies slowly evolved into the idea that actually you can generalize it and talk about general dependencies. So I was one of the people that found out that actually you can first of all express the dependencies in first order logic, and then you can generalize and find out more general formulas which have nice properties. Essentially very closed in spirit to horn formulas and this is why they are so nice and have nice properties.

Then, when Moshe Vardi came along as a student, we formulated the chase (for this general class). Obviously, there were other groups that were working in a similar direction. Fagin was

[1] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. ACM Transactions on Database Systems 4(4):455-469, 1979.

working on data dependencies in a very similar direction. But, for me, this was an important period. On one side, I was lonely, but on the other hand, I managed to come up with a nice idea and the chase, and the paper I authored with Moshe Vardi[2] at the time about the properties of the chase, I think was important. I remember it very fondly, and I think other people have found it as a very useful paper. It provides the fundamental properties of the chase in the general framework. I am very happy to see that now the chase and data dependencies are being used in other contexts. For example, they are being used by Val Tannen and his students for query optimization, and more recently for the work in IBM Almaden on data exchange, which seems to attract a lot of attention and I think it shows that data dependencies are, in some sense, a general idea that can be used in many different directions.

So that's the CLIO and the CLIOLLO project?

That's the Clio project, and I understand that IBM was planning to come up with a product Cliollo, for data exchange, which essentially is based on these ideas. So this is very satisfactory for me.

Among other potential major results of the past 40 years, did you want to say anything about schema design?

Well, schema design was actually my first work on database theory. Historically, I finished my PhD in Jerusalem. It was on automata theory, like almost everybody else at the time, because Jerusalem had the world experts on automata theory at the time, like Shamir and Rabin. So I worked on automata theory. Then I came to Toronto as a post-doc, and I shared the room with somebody named Phil Bernstein, who was a post-doc in Toronto. He stayed in Toronto for another year as a postdoc, we shared the room, and my first step was to read his thesis. So I was introduced to data dependencies, then we had some discussions about his thesis, about database normalization, and we worked further on the subject. A few months later, I was able to prove some NP completeness results about schema design, which at the time was quite new. I was not the first one. Sylvia Osborne, I think, wrote the first NP completeness results. But of course, NP completeness nowadays is mundane, but at that time, it was still exciting.

Then I moved to Princeton for the second year of my postdoc. I gave a database course, and I talked about database design, and dependencies, and Jeff Ullman got involved, and then Yannakakis, and Dave Maier, and Sagiv, and everybody got involved with data dependencies, so there was a bloom of research a lot of research. For me, this was a very difficult period, I must tell you, because, again, I had a family, I was a young postdoc… I didn't know what I was going to do next year, so personally I was under pressure, and I was not sure that data dependencies was the right subject to work on. And I wasn't sure if, you know, tomorrow I could come up with additional results -- typical problems of a young postdoc. But it turns out that yes, the area bloomed, and there was quite a flurry of activity, and PODS was started after I came back to

2 Catriel Beeri, Moshe Y. Vardi: A Proof Procedure for Data Dependencies. Journal of the ACM 31(4): 718-741, 1984

Jerusalem, by the people that essentially studied in Princeton at the time, and got also interested in database theory. Database theory at that time was dependencies, there was almost nothing else in database theory.

So it must have been hard, because it seems like back then all these people were in Toronto as a real hotbed of database activity, and to a lesser degree, Princeton, and then, from what you said, you went back to the Hebrew University and not only were there not any database people, there weren't very many computer scientists, and of course, we didn't have the web and the internet like we do now.

We did have email. Our department was the first one in Israel to have email connection, when I came back, we already had email in Jerusalem.

So you could collaborate.

So actually, I finished my paper with Jeff Ullman via email, I added some additional results, I wrote them down and I simply emailed them. This was done via email. So email was very useful, because otherwise, the paper would have been much shorter.

Did you want to say anything about datalog, what about all that effort on datalog, magic sets?

Maybe we should try to go chronologically, because we talked about my first visit to the States, which was in Toronto and Princeton, and then I came back, and of course, ok, I worked on dependencies. But, you know, people worked on other subjects, and I kept looking at what Phil Bernstein was doing, and he was working on something which I never knew about, which was transaction management. I didn't know what it was. And again, being in Jerusalem, I mean, there was almost no way I could start working on it. I read the papers, but if you are not at the center of action, and you don't talk to people, then you don't really know what are the interesting problems.

So, my first sabbatical was in Harvard, with Phil Bernstein. This was the time when I got introduced to transaction management, and I did some work with Phil, and Nathan Goodman about transaction management. And another one of my very favorite papers[3]… I actually wrote only 4 or 5 papers about transaction management, but the first one was the outcome of this year of work. It is the J. of ACM paper, a very long one, very difficult. I think that maybe not too many people have read the paper, but it's a fundamental one, in terms of providing the theory of how you prove correctness of, let's say, non-simple transaction systems. In other words, systems which have layers or which are complex, how can you go about proving their correctness? For me this was also very interesting, and in a completely different direction, completely different -- another one of my very favorite papers. I took a long time to get it done. It took a long time to get it accepted by J. of ACM, but I still think it is a very good piece of work.

[3] Catriel Beeri, Philip A. Bernstein, Nathan Goodman: A model for concurrency in nested transactions systems. Journal of the ACM 36(2): 230-269 (1989).

Then I went back to Jerusalem, and after a few more years, I went to another sabbatical to MCC, and this was about a year after the first paper on datalog by Jeff Ullman and Yehoshua Sagiv appeared, about magic sets. The word magic sets actually appeared first in their paper in '86. And I came to MCC, and that was also a very intense, and a very satisfying piece of work. The language was LDL: logic data language[4]. It contained sets, and negations, and of course, if you have sets and negations, you can express paradoxes. So the problem was how to come up with satisfactory semantics, and I think in the end, we managed to come up with a nice semantics based on stratification. So it's another nice piece of work, and with it some other nice papers in this period, it was the MCC team, and also Paris Kanellakis came down and we did some work together.

So it seems in this story, the way you are telling it, sabbaticals played a really important role.

For me, yes. Every time I went on a sabbatical, I was lucky enough to get into an environment where there were new problems, and people were working on systems, and theory, and this interaction in systems and theory, and a group of people working on a new subject was important. Sitting alone and giving lectures in a university without interacting with a lot of people is not a good way of finding new problems or new areas.

Yes, it seems a lot of people now file for a sabbatical but then they don't actually go anywhere, they say the kids have to stay in school, or whatever, and they don't actually leave their home institution. Does that matter now, now that we have such good internet?

I think of course the internet has changed a lot, I mean, first of all, it is much easier to get papers. At that time, if you really wanted to look at a paper, you needed either a personal contact, or to be on the mailing list of an institute, and even then, it took time before you got the paper. Even if you get the paper, say you read the paper, the paper is not the full interaction with people. There is a lot of difference between reading a paper and talking to people in the corridor or room, and then being involved in the process of asking questions and getting answers, and trying to find out what is it all about. It is completely different. So actually, when I went to Harvard, I went with my family. But when I went to MCC, I went alone. My wife and the kids stayed in Jerusalem. My wife came for a short visit for a couple of weeks, but most of the time I was alone. And since then, in all of my sabbaticals, I was alone. So instead of taking full year sabbaticals, I took 6 months, or 3 months. So sabbaticals, to me, were always important, because they gave me the opportunity to visit people, to see a different approach, new problems, talk to people, talk to students, and develop. I am not sure I could have developed the same if I had just stayed in Jerusalem and tried to read papers to see what is going on. I think sabbaticals are important.

[4] Catriel Beeri, Shamim A. Naqvi, Oded Shmueli, Shalom Tsur: Set Constructors in a Logic Database Language. Journal of Logic Programming, 10(3&4): 181-232 (1991).

Thinking about another aspect of technology, I understand you collect music, and that has changed a lot.

Yes. I am not sure I am a serious collector, but in all my sabbaticals, in all my visits to the United States, or when I come to conferences, until a couple of years ago, I always go out to find music stores and try to find records that I don't have, or sometimes records that I have recommendations for. Now it is of course CDs. It has always been CDs in the last 20 years. So I remember there was the PODS conference in San Diego a few years ago, and I took Tova Milo along and we went to a record store because I had the recommendation for a certain set of CDs of Haydn which I was looking for and I couldn't find in Israel.

Can't you just order everything from Amazon?

Nowadays, yes, but, Amazon didn't exist 10-15 years ago. Nowadays, everything has changed of course. I understand that now it is very difficult to find in New York a store that sells classical music. Somebody told me this yesterday. I came to New York, and all the record stores were closed because of Tower, now Tower is closing down[5], and you can't find real classical records anymore.

I didn't realize Tower was closing down.

I don't know if this is true or not, but obviously the world is changing, but you know, I remember that I used to go out in different small communities and towns and look for record stores, and you could find, you know, usually you don't find a lot of classical music in the suburban music stores, but you find a little bit. So my first Jascha Heifetz, I found in such a music store, in New Jersey. So I have a large collection. It is limited, I cannot afford to buy much more. I don't have place.

Space limited.

Yes.

So as soon as they change the technology to a smaller form factor, then you can expand again.

Then I can expand, yes.

Do you have any words of advice for fledgling or midcareer database researchers?

Well, I think there are two advices. First of all, be open. Look out in different directions, and talk to people. Try to see what people are doing, and what is bothering people, and maybe you can find some model or answer or formulation that will make life easier and of course advance your research credit. I think I've always benefited from being involved with other people and trying to

[5] Tower used to be a retail music chain based in California. It is currently an international franchise and an online record store.

understand what was bothering them. In other words, I never thought that theory for theory is the right approach. Of course, there are people who do theory for theory who are very successful, but I thought theory for helping formalize what other people are doing is the interesting part.

Among all your past research, do you have a favorite piece of work? I guess you have mentioned several favorite pieces of work, but maybe there is one that we haven't already talked about.

I mentioned a few, but every time I think about it, there is another one. Of course, the paper about transaction management in J. ACM is one that I like very much. There were quite a few. I mean, there was a paper about magic sets with Raghu Ramakrishnan[6] that at the time was very nice, a very nice piece of work. There is another paper with Raghu that was never published, that I did when I visited him in Wisconsin. Because they wanted just a little more, and just a little more, and when I actually did this, well, the paper just never got published. So this is one direction. Then the LDL paper was a nice paper at the time. The LDL language was actually used by Dick Tsur and others for many years to come, and this paper about the language and the semantics was nice. I mentioned the dependencies, the chase the paper about the properties of the chase was a nice piece of work that I really liked, and it seems like some people actually like it even today. So there are quite a few.

If you magically had enough extra time to do one additional thing at work that you are not doing now, what would it be?

That is difficult. I guess I would like to do a good piece of research in programming languages. Because I have always been reading programming language theory papers, but I somehow never got around to actually solve an interesting problem in programming languages. So I teach programming languages, and I read, but this one last step of doing actual research and providing a good solution to some interesting problem is something I never got around to do. So I would like to do that.

If you could change one thing about yourself as a computer science researcher, what would it be?

I guess I would have tried to be a much better programmer earlier.

Why?

Well, I am not sure I have a good reason. I always feel that my students know programming so well, so they can develop all sorts of interesting programming systems, and I cannot. So I do theory. Is it because I am good at theory, or because I cannot do implementations? It is not clear. I always think that maybe next year I will sit down and devote a few months to actually just writing programs, and see what happens. Then I can supervise students doing more interesting, maybe systems work. But I am not sure this is a good reason. I think altogether, I have done

[6] Catriel Beeri, Raghu Ramakrishnan: On the Power of Magic. PODS 1987: 269-284.

some good theory, and I know theoreticians who don't know how to program in even one language.

Well, we won't name their names here, they are to remain anonymous!

No! But they are very good, so it is nothing to be ashamed of. You cannot really be a professor and a good programmer, and I think this is the truth. You cannot be a researcher, and a theoretician, and a professor, and a teacher, and devote 5 hours a day to programming.

That's true, it is very time consuming.

It is time consuming. So there is a trade-off.

Well, thank you very much for talking with me today.

You're welcome.

Query Languages for Graph Databases

Peter T. Wood
Department of Computer Science and Information Systems, Birkbeck, University of London

ptw@dcs.bbk.ac.uk

ABSTRACT

Query languages for graph databases started to be investigated some 25 years ago. With much current data, such as linked data on the Web and social network data, being graph-structured, there has been a recent resurgence in interest in graph query languages. We provide a brief survey of many of the graph query languages that have been proposed, focussing on the core functionality provided in these languages. We also consider issues such as expressive power and the computational complexity of query evaluation.

1. INTRODUCTION

Graphs are widely used for representing data, with the result that a number of query languages for graphs have been proposed over the past few decades. In the 1980s motivating applications came from areas such as hypertext systems [17, 63]. When semi-structured data [2, 12] and object databases [34] became prominent in the 1990s, these provided fruitful areas for the study of graph models and query languages. In the last decade, the semantic web [9, 57] and social networks [5, 24, 60, 61] have taken over as key areas amenable to graph-based approaches. Further application areas for graph querying include transportation networks [11], semantic associations as part of criminal investigations [62] (also called link analysis), biological networks [46, 47, 48], program analysis [50], workflow and data provenance [6, 39].

Each of the above application areas has its own requirements in terms of an appropriate graph data model. In its simplest form a graph G is a pair (V, E), where V is a finite set of nodes and E is a finite set of edges connecting pairs of nodes. Of course, edges can be directed or undirected, although we will consider only the directed case here (which is more general). In most applications it is also the case that edges are labelled in some way, sometimes with sets of attribute-value pairs. Similarly, in general nodes may be labelled with sets of attribute-value pairs. However, we will mostly limit our discussion to graphs in which each node is identified by a distinct label (identifier) and each directed edge is labelled with a symbol drawn from some finite alphabet Σ; hence $E \subseteq V \times \Sigma \times V$.

Some application areas require graph structures that are more elaborate than the simple model described above. For example, hypergraphs have been used to model hypertext [63], while the hypernode model [58] allows for nodes that can themselves comprise graphs. In contrast, the so-called blobs of the Hy$^+$ system [19] comprise sets of nodes and share some similarities with the blobs of higraphs [35]. In addition, a number of graph data models require that each graph conforms to a *schema*. However, for the purposes of this paper, we will assume only the simple model described above; for details on more elaborate graph data models, we refer the reader to the survey by Angles and Gutiérrez [7].

Figure 1 shows an example of a graph G conforming to our simple definition, inspired by an example from NAGA [43, 64]. Node labels in G denote names of authors, literary prizes and locations. Edge labels denote the $hasWon$ relationship between authors and prizes (abbreviated w), the $bornIn$ relationship between authors and places (abbreviated b), the $livesIn$ relationship between authors and places (abbreviated i), and the $locatedIn$ relationship between places. Note that there can be paths comprising $locatedIn$ edges: for example, Bacchus Marsh is a town $locatedIn$ the state of Victoria which is $locatedIn$ the country Australia.

A typical query Q on graph G might ask to find authors who have won both the Booker and Nobel prizes—a simple *conjunctive query* (CQ) returning a set of nodes as answer. Query Q might be expressed using the following syntax

$$ans(x) \leftarrow (x, hasWon, Nobel), (x, hasWon, Booker)$$

where x is interpreted as a node variable, while

Database Principles Column. Column editor: Pablo Barceló, Department of Computer Science, University of Chile. E-mail: pbarcelo@dcc.uchile.cl.

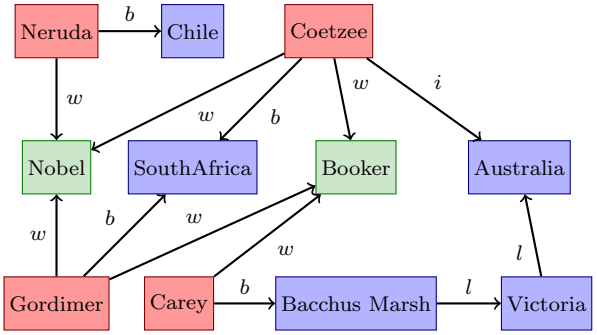


Figure 1: A graph of authors, prizes they have won, and places where they were born.

hasWon, *Nobel* and *Booker* are constants. This syntax is reminiscent of Datalog, except that atoms in the body do not have predicate names since only a single graph is being queried; indeed, the atoms are similar to the *triple patterns* used in SPARQL [36], the W3C query language for RDF [44].

It is common in querying graphs that users may want to find pairs (x, y) of nodes such that there is a path from x to y whose sequence of edge labels matches some pattern. One way of specifying such a pattern is by using a *regular expression* defined over the alphabet of edge labels [52]. Such a query is called a *regular path query* (RPQ). So, using the example of Figure 1, an RPQ using the regular expression $citizenOf \mid ((bornIn \mid livesIn) \cdot locatedIn^*)$ asks for pairs of author x and place y such that x is a citizen of y or was born in or lives in y, or where x was born in or lives in some place which is connected to y by a sequence of any number of *locatedIn* relationships.

CQs and RPQs can be combined to form *conjunctive regular path queries* (CRPQs). For example, the following query Q adds the conjuncts from the example CQ to those of the RPQ as follows:

$$ans(x, y) \leftarrow (x, hasWon, Nobel), (x, hasWon, Booker) \\ (x, (citizenOf \mid ((bornIn \mid livesIn) \cdot locatedIn^*)), y)$$

CRPQs formed the basis of the languages **G** [21] and GraphLog [17], although those languages used a syntax of graph patterns. Figure 2 shows how the CRPQ Q above would be expressed in GraphLog, where there is an obvious mapping between the edges in the graph pattern and the atoms in the body of Q. The thick edge in Figure 2 is called the *distinguished* edge, representing edges which occur in the answer of the query; hence, it corresponds to the head of Q.

CRPQs were much studied with respect to querying semistructured data in languages such as Lorel

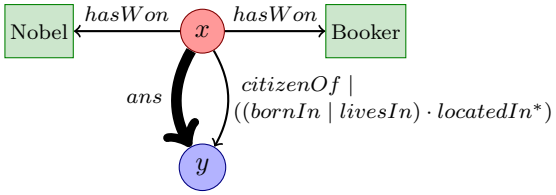


Figure 2: Query to find places related to authors who have won both the Nobel and Booker prizes.

[2], StruQL [26] and UnQL [13], as well as in terms of query containment [16, 28], query rewriting [15] and so on. CRPQs reappeared more recently in NAGA [43], for example, in a form similar to that in Figure 2. The ability to query paths using regular expressions, and hence provide the functionality of CRPQs, has only very recently been introduced in SPARQL 1.1 [36].

However, for a number of problems arising in graph querying CRPQs are insufficiently powerful [10]. These problems include comparing semantic associations in RDF graphs [8], comparing biological sequences [31], and so on. In such settings, we want to be able to express *relations* among paths. For example, the following query Q finds entities x and y such that the same sequence of edge labels π connects x and y as connects *Coetzee* and y:

$$(x, y) \leftarrow (Coetzee, \pi, y), (x, \pi, y), \Sigma^*(\pi) \quad (1)$$

Here, π is a *path variable*, and Σ^* denotes any sequence of edge labels. Q is an example of an *extended* conjunctive regular path query (ECRPQ), as proposed in [10].

We might also want to include paths themselves in the output of a query. This has been proposed, for example, as an extension to SPARQL [45], and is also provided by ECRPQs; to return the paths as part of the answer to query Q above, one simply includes the path variable π in the head. Of course, if there are cycles in the input graph, the answer to a query may be infinite. In such cases a compact representation of the set of answers to an ECRPQ can be returned in the form of an automaton [10]. ECRPQs are described in more detail in Section 3.3.

Requirements arising from querying and analysing social networks bring the need for further capabilities to be provided by graph query languages [5, 24, 60, 61]. In particular, aggregation functions play an essential role in network analysis, while the ability to transform networks by creating new nodes based on (aggregations of) sets of existing nodes is also crucial. We discuss these requirements further, and provide examples, in Sections 3.4 and 3.5.

With the proliferation of data on the Web (e.g., in the form of linked data), it is less likely that users will be familiar with the terms and structure used, which in any case will also be more heterogeneous. In such situations, queries that permit flexible [42, 51] or approximate matching [30, 40] of data may be helpful. We consider this further in Section 3.6.

After a brief survey of many graph query languages in the next section, we focus on the core functionality provided by such languages in Section 3. This section covers subgraph matching (Section 3.1), finding nodes connected by paths (Section 3.2), comparing and returning paths (Section 3.3), aggregation (Section 3.4), node creation (Section 3.5), and approximate matching and ranking (Section 3.6). Section 4 covers the expressive power of languages and the computational complexity of query evaluation. We conclude the paper in Section 5. We do not cover a number of other topics of interest such as query containment or query optimisation or evaluation in general.

2. A BRIEF SURVEY

In this section we give a brief overview of some of the graph query languages developed over the past 25 years or so. In particular, we highlight the different syntax used by various languages, as well as their proposed area of application. Section 3 discusses the functionality underlying these languages in more detail, while the expressive power and complexity of evaluating queries in some of the languages is presented in Section 4.

We have already mentioned the query languages **G** [21] and GraphLog [17]. The data model used by these languages is that of a labelled, directed graph. In G, a query is a set of pairs of graphs, each pair comprising a pattern graph and a summary graph. This pair of graphs essentially represents a CRPQ, with a set of such pairs being interpreted as disjunction. GraphLog replaced the summary graph with a distinguished edge, as shown in Figure 2. GraphLog also added edge inversion, negation and aggregation functions (Section 3.4), while defining a semantics different from that of **G**. The semantics of **G** was defined in terms of matching *simple* paths in the graph being queried (Section 3.2), whereas the meaning of a GraphLog query was given by the meaning of the stratified Datalog program to which it was translated (examples are given in Sections 3.2 and 3.4).

Other early graph query languages include GRAM [4] and GraphDB [32]. The data models of both require the presence of a graph schema. Both provide regular expressions defined over alternating sequences of node and edge types. GraphDB includes object-oriented features such as classes for nodes and edges, as well as paths. The intended area of application for GRAM was Hypertext, while that for GraphDB was spatial networks such as transportation systems. As a result, GraphDB provides built-in operators such as one for shortest path.

GOOD is another graph query language based on an object-oriented model [33, 34]. GOOD's querying mechanism is via graph transformations: node addition/deletion and edge addition/deletion. Also provided is an abstraction mechanism to group objects by means of their properties, as well as methods for defining sequences of operations. GOOD gave rise to a number of successor languages, such as G-Log [56] and the update language GUL [38].

A number of query languages were developed to query graphs represented in the Object Exchange Model (OEM) [55] or one of its variants/derivatives. OEM was developed to model semistructured data which had no predefined schema and could be heterogeneous. The Lore (Lightweight Object Repository) graph data model and its associated query language Lorel [2] distinguish two types of nodes: complex objects and atomic objects (values) which have no outgoing edges. A graph must have a number of named nodes (or entry points), and every node must be reachable from a named node.

In common with many graph query languages, Lorel uses a syntax based on OQL, allowing regular expressions over edge labels. A distinctive feature of Lorel is the availability of path variables.

Say we wish to formulate a Lorel query Q equivalent to the ECRPQ shown in (1). In Lore, each node has an oid rather than a label as in Figure 1, so authors' names, for example, would be represented by separate atomic objects connected to author nodes by an edge labelled *name*, say. We also assume that *Winners* is a named entry point, with edges labelled *author* to author nodes. Then Q can be written as

select X, Y
from *Winners.author* A, *Winners.author* X
$A.\#@P.Y$, $X.\#@Q.Y$
where $A.name$ = *'Coetzee'*
and *path-of*(P) = *path-of*(Q)

where # denotes a path of any length, so is equivalent to the regular expression Σ^* used earlier. $@P$ binds the path (of oids and labels) to variable P, and *path-of* returns a sequence of edge labels.

STRUDEL is another system whose data model is based on the OEM data model [26]. Its intented area of application is the implementation of data-intensive web sites, whose content and structure is specified using the query language STRUQL. The

language is compositional; the result of a query on a site graph is another site graph. Once again, regular path expressions are used, but because there is a need to create graphs corresponding to web sites, new constructs are needed. These include the `link` clause which creates a new graph from existing graphs, using Skolem terms for new nodes.

UnQL is a functional query language for semistructured data based on structural recursion [13]. The data model used somewhat different to that of OEM, being value-based rather than object-based. UNQL queries are translated to an internal algebra, UnCAL, which allows for optimisation. Once again, regular path patterns are provided, but the functional nature of UnQL means that graphs can be constructed, using data constructors, rather than only queried. UnQL's data model includes special symbols called *markers*, which are related to object identifiers in models such as OEM, except that not every node in a graph has to have a marker. Each graph also has certain nodes designated as inputs and certain nodes designated as outputs.

YAGO/NAGA combines database and information retrieval techniques to provide a semantic search engine for web derived knowledge [64]. The NAGA data model is a directed, weighted multigraph in which nodes represent entities, edges represent relationships, and weights represent confidence of extracted facts. A query is a connected, directed graph in which each edge is labelled with a regular expression over edge labels or a variable or the *connect* keyword (similar to Figure 2). A query using *connect* returns the paths connecting the corresponding nodes. Answers to queries are ranked in terms of informativeness, confidence and compactness (e.g., short paths rank higher than long paths).

SocialScope [5] aims to provide information discovery and presentation from social content sites such as Yahoo! Travel. To do this, it proposes a uniform algebraic framework operating on the social content graph, a graph in which both nodes and edges have attributes. The algebra provides operations of node selection, edge selection, graph union, intersection and difference, graph composition and semi-join, and aggregation functions for node aggregation and edge aggregation.

Other query languages for social networks include SoSQL [60], BiQL [24] and SNQL [61]. The two basic query structures in SoSQL are paths and groups (sets of nodes). Paths can have predicates and aggregate functions applied to them. Path predicates can include path operators such as *all* or *at most* n, specifying that all or at most n nodes or edges satisfy a given predicate. Groups can also have aggregation operators applied to them. BiQL [24] uses an SQL-like syntax to query and transform networks. New networks are formed using a `CREATE` clause. Aggregation functions can be used in the `WHERE` clause to restrict results, as well as in the `SELECT` clause to define new attribute values. SNQL [61] uses a data model comprising actors, relationships and attribute values (all represented as nodes), with edges associating attribute values with actors or relationships and associating actors with the relationships in which they participate. The query language is based on GraphLog, adding Skolem functions in order to create new nodes in the output. These new nodes are based on grouping or aggregating nodes or attribute values in the input (see Section 3.5).

3. QUERY LANGUAGE FUNCTIONALITY

In this section, we focus on the functionality provided by typical graph query languages. The following subsections will consider functionality in terms of the following broad categories: subgraph matching, finding nodes connected by paths, comparing and returning paths, aggregation, node creation, and approximate matching and ranking. Of course, many languages offer operations such as union (disjunction), composition and negation of queries, but we will not cover these separately.

We start with some general notation and definitions. Let $G = (V, E)$ be a graph as defined in Section 1. Given a query expression Q and a graph G, the evaluation of Q on G is denoted $Q(G)$.

Let us call the following question the *query evaluation problem* (QEP). Given a query expression Q and a graph G, is $Q(G)$ non-empty? As usual, one can consider the complexity of this problem by possibly fixing one of the two inputs. *Combined complexity* corresponds to when both Q and G are part of the input. *Query complexity* is when the input is Q, with G being fixed, while *data complexity* is when the input is G, with Q being fixed. We often consider data complexity to be the most relevant measure since graphs are assumed to be large and query expressions short.

3.1 Subgraph matching

In some sense, the simplest form of graph query supported by all languages is one which finds subgraphs within a graph. This corresponds to a *conjunctive query* (CQ). Let us fix a countable set of *node* variables (typically denoted by $x, y, z, \ldots$). A

We will from now on not usually distinguish between an expression in a query language and the query (function) it denotes, simply using the term "query" for both.

conjunctive query (CQ) Q over a finite alphabet Σ is an expression of the form:

$$ans(z_1, \ldots, z_n) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, a_i, y_i) \qquad (2)$$

such that $m > 0$, each x_i and y_i is a node variable or constant ($1 \leq i \leq m$), each $a_i \in \Sigma$ ($1 \leq i \leq m$), and each z_i is some x_j or y_j ($1 \leq i \leq n, 1 \leq j \leq m$). The atom $ans(z_1, \ldots, z_n)$ is the *head* of the query, while the expression on the right of the arrow is its *body*. The query Q is *Boolean* if its head is of the form $ans()$, i.e. $n = 0$.

Let $\bar{x} = (x_1, \ldots, x_m)$, $\bar{y} = (y_1, \ldots, y_m)$ and $\bar{z} = (z_1, \ldots, z_n)$. The semantics of CQs Q of the form (2) are defined as follows. Let σ be a mapping from $\bar{x}, \bar{y}$ to the set of nodes of a graph $G = (V, E)$ which is the identity on constants. We define a relation $(G, \sigma) \models Q$ which holds iff $(\sigma(x_i), a_i, \sigma(y_i)) \in E$, for $1 \leq i \leq m$. Then $Q(G)$ is the set of tuples $\sigma(\bar{z})$ such that $(G, \sigma,) \models Q$. If Q is Boolean, we let $Q(G)$ be true iff $(G, \sigma) \models Q$ for some σ.

Using the example graph G from Figure 1, the following CQ finds authors born in South Africa who have won both the Nobel and Booker prizes:

$$\begin{aligned} ans(x) \leftarrow\ & (x, hasWon, Nobel), \\ & (x, hasWon, Booker), \\ & (x, bornIn, SouthAfrica) \end{aligned}$$

Each CQ of the form shown in (2) is formulated with respect to a single graph and returns a set of bindings for each node variable mentioned in the head of the query. However, in some application areas, the database to be queried comprises a *set* of graphs, and the answer to a query is the *subset* of graphs in which a match is found (e.g., in biological applications). Other languages allow single or multiple graphs to be queried and return the set of matching *subgraphs* [37, 43].

Although CQs are in some sense the simplest form of graph queries, they have been the subject of much study, particularly in terms of finding efficient ways of evaluating them on large graphs. This is because the combined complexity of the QEP for CQs is the same as the problem of subgraph ismorphism, which is well-known to be NP-complete. Because of this, Fan et al. have been investigating an alternative semantics for graph pattern matching based on graph simulation [25].

3.2 Finding nodes connected by paths

Let $G = (V, E)$ be a graph over alphabet Σ, with $v_0, v_m \in V$. A *path* ρ between nodes v_0 and v_m in G is a sequence $v_0 a_0 v_1 a_1 v_2 \cdots v_{m-1} a_{m-1} v_m$, where $m \geq 0$, $v_i \in V$ ($1 \leq i \leq m$), $a_i \in \Sigma$ ($1 \leq i < m$), and $(v_i, a_i, v_{i+1}) \in E$ ($1 \leq i < m$). The *label* of such a path ρ, denoted by $\lambda(\rho)$, is the string $a_0 \cdots a_{m-1} \in \Sigma^*$. The length of ρ is m. We also define the empty path as (v, ε, v) for each $v \in V$; the label of such a path is the empty string ε.

Regular path queries Determining reachability between nodes in a graph is a querying mechanism found in most graph query languages. The class of *regular path queries* [15, 21, 49, 52] provides queries which return all pairs of nodes in a graph connected by a path conforming to some regular expression. A *regular path query* (RPQ) Q is an expression of the form

$$ans(x, y) \leftarrow (x, r, y) \qquad (3)$$

where x and y are node variables, and r is a regular expression over Σ. Here we use $|$ for alternation (disjunction) and $\cdot$ for concatenation. We also allow the shorthands of r^+ for $(r \cdot r^*)$, $r?$ for $(r|\epsilon)$, and Σ for $(a_1| \cdots |a_n)$. We may also use a^- to match an edge labelled a in the *reverse* direction, i.e., from head to tail rather than from tail to head [16]. For example, the regular expression $citizenOf \mid ((bornIn \mid livesIn) \cdot locatedIn^*)$ is used in the example query in Figure 2.

Let G be a graph, r be a regular expression, and ρ be a path in G. Path ρ *satisfies* r if $\lambda(\rho) \in L(r)$, the language denoted by r. Given an RPQ Q of the form given in (3), the answer of Q on G, denoted $Q(G)$, is the set of all pairs of nodes (x, y) in G such there is a path from x to y which satisfies r.

The Regular Path Problem is, given a query Q of the form given in (3), a graph G and a pair of nodes x and y, is $(x, y) \in Q(G)$? It is well known that this problem can be solved efficiently [52]. One algorithm is as follows: (i) construct a nondeterministic finite automaton (NFA) M_r, with initial state s_0 and final state s_f, accepting $L(r)$; (ii) consider G as an NFA with initial state x and final state y; (iii) form the product automaton $M_r \times G$ of M_r and G; and (iv) determine whether there is a path from (s_0, x) to (s_f, y) in $M_r \times G$. Each step of this algorithm can be performed in Ptime, so the Regular Path Problem has Ptime combined complexity.

Alternatively, we can translate Q into a set of Datalog rules. So if Q uses the regular expression $citizenOf \mid ((bornIn \mid livesIn) \cdot locatedIn^*)$ from Figure 2, the translation might be as follows:

$$\begin{aligned} ans(x, y) &\leftarrow citizenOf(x, y) \\ ans(x, y) &\leftarrow assoc(x, y) \\ ans(x, y) &\leftarrow assoc(x, z), partOf(z, y) \\ assoc(x, y) &\leftarrow bornIn(x, y) \\ assoc(x, y) &\leftarrow livesIn(x, y) \\ partOf(x, x) &\leftarrow locatedIn(x, x) \\ partOf(x, y) &\leftarrow locatedIn(x, z), partOf(z, y) \end{aligned}$$

As an alternative to the semantics defined above, we might instead want to match only *simple* paths in G that satisfy the regular expression r. A path ρ is *simple* if no node is repeated on ρ. The Regular Simple Path Problem can then be stated as, given a graph G, a pair of nodes x and y in G and a regular expression r, is there a *simple* path from x to y satisfying r? It turns out, however, that the Regular Simple Path Problem is NP-complete, even for fixed regular expressions [52].

Conjunctive regular path queries Extending regular path queries by allowing conjunctions of atoms yields the class of *conjunctive* regular path queries [16, 28]. A *conjunctive regular path query* (CRPQ) Q over Σ is an expression of the form

$$ans(z_1, \ldots, z_n) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, r_i, y_i) \qquad (4)$$

in which all variables are as for a CQ, except that each r_i is now a regular expression over Σ. The query of Figure 2 corresponds to a CRPQ.

Let $\bar{x}$, $\bar{y}$ and $\bar{z}$ be defined as for CQs, and $G = (V, E)$ be a graph. The semantics of CRPQs Q of the form (4) are defined analogously to that for CQs, with σ being a mapping from $\bar{x}, \bar{y}$ to V. The relation $(G, \sigma) \models Q$ holds iff, for $1 \leq i \leq m$, there exists a path ρ_i in G from $\sigma(x_i)$ to $\sigma(y_i)$ such that $\lambda(\rho_i) \in L(r_i)$. Now $Q(G)$ is defined as for CQs.

3.3 Comparing and returning paths

As suggested in Section 1, there are a number in situations in which we may want to specify relations between paths as well as to have the actual path(s) connecting two nodes returned as query answers, whether to find connections in linked data on the web (DBPedia, Freebase), for analysis in social or other networks, or to determine data provenance.

Providing both of these capabilities gives rise to the class of *extended CRQPs* (ECRPQs), introduced in [10] from where most of the material in this subsection is derived. ECRPQs extend the class of CRPQs in two ways. Firstly, they allow *free path variables* in the heads of queries. Secondly, they permit checking *relations on sets of paths* in the bodies of queries, rather than simply conformance of individual paths to regular languages.

We first define the notion of *regular* relations over Σ, which are used in ECRPQs. Let $\perp$ be a symbol not in Σ. We denote the extended alphabet $(\Sigma \cup \{\perp\})$ by $\Sigma_\perp$. Let $\bar{s} = (s_1, \ldots, s_n)$ be an n-tuple of strings over alphabet Σ. We construct a string $[\bar{s}]$ over alphabet $(\Sigma_\perp)^n$, whose length is the maximum of the s_j's, and whose i-th symbol is a tuple $(c_1, \ldots, c_n)$, where each c_k is the i-th symbol of s_k, if the length of s_k is at least i, or $\perp$ otherwise. In other words, we pad shorter strings with the symbol $\perp$, and then view the n strings as one string over the alphabet of n-tuples of letters. An n-ary relation S on Σ^* is *regular* if the set $\{[\bar{s}] \mid \bar{s} \in S\}$ of strings over alphabet $(\Sigma_\perp)^n$ is regular (i.e., accepted by an automaton over $(\Sigma_\perp)^n$, or given by a regular expression over $(\Sigma_\perp)^n$). We shall often use the same letter for both a regular expression over $(\Sigma_\perp)^n$ and the relation over Σ^* it denotes, as doing so will not lead to ambiguity.

In additional to the set of node variables defined for CRPQs, we now also fix a countable set of *path* variables (denoted by $\pi, \omega, \chi, \ldots$). An *extended conjunctive regular path query* (ECRPQ) Q over Σ is an expression of the form:

$$ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\bar{\omega}_j), \quad (5)$$

such that

(i) $m > 0$, $t \geq 0$,

(ii) each R_j is a regular expression that defines a regular relation over Σ,

(iii) $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{y} = (y_1, \ldots, y_m)$ are tuples of node variables,

(iv) $\bar{\pi} = (\pi_1, \ldots, \pi_m)$ is a tuple of distinct path variables,

(v) $\{\bar{\omega}_1, \ldots, \bar{\omega}_t\}$ are distinct tuples of path variables, such that each $\bar{\omega}_j$ is a tuple of variables from $\bar{\pi}$, of the same arity as R_j,

(vi) $\bar{z}$ is a tuple of node variables from $\bar{x}, \bar{y}$, and

(vii) $\bar{\chi}$ is a tuple of path variables from those in $\bar{\pi}$.

The semantics of ECRPQs is defined by a natural extension of the semantics of CRPQs. For an ECRPQ Q of the form (5), a graph $G = (V, E)$ and mappings σ from node variables to V and μ from path variables to paths, we write $(G, \sigma, \mu) \models Q$ if

- $\mu(\pi_i)$ is a path in G from $\sigma(x_i)$ to $\sigma(y_i)$, for $1 \leq i \leq m$, and
- for each $\bar{\omega}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$, the tuple of strings consisting of labels of $\mu(\pi_{j_1}), \ldots, \mu(\pi_{j_k})$ belongs to the relation R_j.

The answer of Q on G is defined as

$$Q(G) \;=\; \{(\sigma(\bar{z}), \mu(\bar{\chi})) \mid (G, \sigma, \mu) \models Q\}.$$

We now present some examples, taken from [10]. In a query language for RDF/S introduced in [8], paths can be compared based on specific *semantic associations*. Edges correspond to RDF properties and paths to property sequences. A property a can

be declared to be a subproperty of property b, which we denote by $a \prec b$. Two property sequences u and v are called ρ-isomorphic if and only if $u = u_1, \ldots, u_n$ and $v = v_1, \ldots, v_n$, for some n, and $u_i \prec v_i$ or $v_i \prec u_i$ for every $i \leq n$ [8]. Nodes x and y are called ρ-isoAssociated iff x and y are the origins of two ρ-isomorphic property sequences.

Finding ρ-isoAssociated nodes cannot be expressed using a CRPQ, not least because doing so requires checking that two paths are of equal length. However, pairs of ρ-isomorphic sequences can be expressed using the regular relation R given by the expression $\left(\bigcup_{a,b\in\Sigma:\ (a\prec b \vee b \prec a)}(a,b)\right)^*$. An ECRPQ returning pairs of nodes x and y that are ρ-isoAssociated can then be written as follows:

$$ans(x,y) \leftarrow (x,\pi_1,z_1),(y,\pi_2,z_2),R(\pi_1,\pi_2)$$

Path variables in an ECRPQ can also be used to return the actual paths found by the query, a mechanism found in the query languages proposed in [2, 8, 39, 45]. For instance, SPARQLeR [45] introduces path variables and regular expressions into the SPARQL query language, allowing paths to be output. As an example, the SPARQLeR query that returns every path between the RDF resources r and s, provided the path includes the resource e, can be expressed by the ECRPQ

$$ans(\pi_1,\pi_2) \leftarrow (r,\pi_1,e),(e,\pi_2,s)$$

where π_1 and π_2 are the actual paths matched.

Regular expressions with backreferencing (REBRs) [3], as provided by *egrep* and Perl, for example, extend regular expressions by including expressions of the form $(r)\%X$, where r is a regular expression and X is a variable, which binds a string $w \in L(r)$ to X. Subsequent uses of X in the expression then match w. REBRs can denote non-regular languages [3]. Although ECRPQs can mimick some of this functionality over paths in a graph, it was recently shown that ECRPQs cannot express all REBRs [29]. On the other hand, ECRPQs can match patterns, such as $a^n b^n c^n$, where $a, b, c \in \Sigma$ and $n \in \mathbb{N}$, that cannot be denoted by REBRs, by using an equal-length (el) predicate as follows:

$$\begin{aligned} ans(x,y) \leftarrow\ & (x,\pi_1,z_1),(z_1,\pi_2,z_2),(z_2,\pi_3,y), \\ & a^*(\pi_1),b^*(\pi_2),c^*(\pi_3), \\ & \mathrm{el}(\pi_1,\pi_2),\mathrm{el}(\pi_2,\pi_3), \end{aligned}$$

where $\mathrm{el}(\pi,\pi')$ is shorthand for $(\bigcup_{a,b\in\Sigma}(a,b))^*(\pi,\pi')$.

3.4 Aggregation

Determining various properties of graphs requires computation that goes beyond matching and path finding. Such properties range from simple computations to determine the degrees of nodes to more complex ones for computing the eccentricity of a node, the distance between pairs of nodes, or the diameter of a graph. To formulate queries which return the values of such properties, we need aggregation operators such as *count*, *sum*, *min* and *max*.

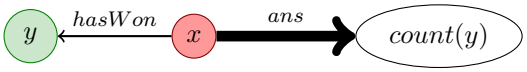


Figure 3: GraphLog query to count the number of prizes for each author.

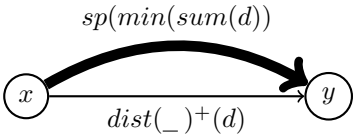


Figure 4: GraphLog shortest path query

Aggregation was available in early graph query languages such as $\mathbf{G}^+$ [22] and GraphLog [20]. It is also a feature of query languages for social networks, such as [24, 60, 61]. GraphLog and SNQL [61] have semantics that are based on Datalog with aggregation. For consistency with the rest of the paper, we will use the language of CRPQs extended with aggregation functions, denoted CRPQ^{agg}, without giving a formal definition.

In GraphLog [20], aggregate terms are allowed in the label of a distinguished edge or distinguished node. The simple query in Figure 3 counts, for each author x, the number of prizes y they have won. The equivalent CRPQ^{agg} is:

$$ans(x, count(y)) \leftarrow (x, hasWon, y)$$

In general, GraphLog and SNQL queries are translated into *recursive* Datalog programs, and combining recursion with aggregation can lead to nontermination. Hence, care is taken to ensure that the recursive rules that are produced perform transitive closure computations over closed semirings [18], such as formed by the operators *min* and *sum* in computing shortest paths, as illustrated below.

Finding the length of the shortest path between each pair of nodes requires that we summarise values (i.e., distances) along a path (by summing them) and then aggregate these summarised values (by finding the minimum). This is what the GraphLog query Q in Figure 4 does. The variable d in Q is a *collecting* variable, *sum* is a *summarising* function, and *min* is used to aggregate the summarised distances. Query evaluation is in PTIME if summarisation and aggregation operators form a closed semiring [18]. Query Q might be translated to the

following Datalog program [20]:

$$\begin{array}{lcl} len(x,x,x,0) & \leftarrow & dist(x,y,l) \\ len(x,x,x,0) & \leftarrow & dist(y,x,l) \\ len(x,z,y,d) & \leftarrow & sp(x,z,s), dist(z,y,l), d = s+l \\ sp(x,y,min(d)) & \leftarrow & len(x,z,y,d) \end{array}$$

Predicate $len(x, z, y, d)$ specifies that there is a path of length $d = s + l$ from x to y via z, where the length of the shortest path from x to z is s and the distance from z to y is l.

As mentioned in Section 2, SocialScope [5] provides node aggregation and edge aggregation. The result of aggregation can be represented as a new attribute of a node or edge in the graph. With the help of a user-defined function for defining similarity between users, [5] shows how aggregate functions can be used to express collaborative filtering on travel recommendations.

3.5 Node creation

In languages such as GraphLog the answer to a query is a graph, where the edges and their labels can be new, but the nodes are drawn from those of the graph being queried. However, in a number of applications, there is a need for the output of a query to contain nodes that were not part of the input. This requirement appears in web site management (and hence in StruQL [26]) and in social network analysis and transformation (and hence in BiQL [24] and SNQL [61]), for example. More general graph creation is also a feature of languages that adopt a functional or algebraic approach such as UnQL [13], GraphQL [37], and GOOD [34].

One approach to supporting node creation, followed in a number of languages, is to adopt a mechanism equivalent to that of Skolem functions, first used for semistructured data in the Mediator Specification Language MSL [54]. In a query language supporting recursion, it is important to ensure that, where possible, node creation and recursion do not combine to yield non-termination.

Recall that BiQL [24] can create new graphs from existing graphs. To do this new object identifiers are needed. They define semantics of their basic language in terms of a translation to Datalog, and then extend this by means of a single Skolem function to represent new object identifiers.

As mentioned in Section 2, SNQL [61] also uses Skolem functions to represent new nodes in the output of a query. Assume that we have a network representing people and the cities in which they live. City names are represented as values of the *livesIn* attribute associated with people. We want to transform this network into one in which cities become actors (rather than values), with *name* and *population* attributes. Although SNQL provides a graph-based syntax, the following Datalog-like program shows a simplification of how such a query might be translated

$$\begin{array}{lcl} ans(f(c), isa, city) & \leftarrow & temp(p,c) \\ ans(f(c), name, c) & \leftarrow & temp(p,c) \\ ans(f(c), population, count(p)) & \leftarrow & temp(p,c) \\ temp(p,c) & \leftarrow & (p, isa, person), (p, livesIn, c) \end{array}$$

Here, f is a Skolem function and *count* is an aggregate function. In general, more than one Skolem function may be needed in an SNQL query.

3.6 Approximate matching and ranking

In many applications, users may not be familiar with the graph structure being queried, its constraints or edge labels. As a result, they may formulate queries which return no answers or fewer answers than they expected. Early work to address such problems when querying semistructured data was done by Kanza and Sagiv [42], who proposed a form of flexible querying based on a notion of homeomorphism between a query and a graph.

In this section, we will consider a more general notion of approximate matching of paths [30, 40], where the results can be ranked in terms of their "closeness" to the original query. Consider a regular path query Q of the form given in (3), using regular expression r. Approximate matching is achieved by applying *edit operations* to $L(r)$. Possible edit operations include insertion, deletion, substitution, transposition and inversion of symbols. For simplicity, we will only consider the first three of these here.

Let $x, y \in \Sigma^*$. Applying an edit operation to x yielding y can be modelled as a binary relation $\rightsquigarrow$ over Σ^* such that $x \rightsquigarrow y$ holds iff there exist $u, v \in \Sigma^*$, $a, b \in \Sigma$, with $a \neq b$, such that one of the following is satisfied:

$$\begin{array}{lll} x = uav, & y = ubv & \text{(substitution)} \\ x = uav, & y = uv & \text{(deletion)} \\ x = uv, & y = ubv & \text{(insertion)} \end{array}$$

Let $\overset{k}{\rightsquigarrow}$ stand for the composition of $\rightsquigarrow$ with itself k times. The *edit distance* $d_e(x, y)$ between x and y is the minimum number k of edit operations such that $x \overset{k}{\rightsquigarrow} y$.

Each operation may have a different cost. In general, different instances of the same operation may have different costs. For example, a user may be prepared to substitute *taxi* by *train* at a cost of one, but *taxi* by *bus* at a cost of two.

More generally, Grahne and Thomo study approximate matching of RPQs in [30], where they

assume that approximations are specified by means of a *weighted regular transducer*. Such a transducer can be represented by a regular expression defined over triples of the form (a, k, b), which specify that a can be replaced by b with cost k.

A *weighted transducer* $T = (S_T, \Sigma, \sigma_T, S_{0_T}, F_T)$ comprises a finite set of states S_T, an input/output alphabet Σ, a set of initial states S_{0_T}, a set of final states F_T, and a transition relation $\sigma_T \subseteq S_T \times \Sigma \times \mathbb{N} \times \Sigma \times S_T$. A transition (s, a, k, b, t) specifies that if the transducer is in state s and reads symbol a, it outputs symbol b at cost k and moves to state t.

As stated in Section 3.2, we can construct an NFA M_r from the regular expression r in query Q and can consider graph G as an NFA as well. Now we can form the product automaton $M_r \times T \times G$ (see [30] for details). Then $(a, b, k) \in ans_T(Q, G)$ iff the shortest path from an initial state $(\_, \_, a)$ to a final state $(\_, \_, b)$ in $M_r \times T \times G$ has cost k. As noted in [30], if we are interested in nodes reachable from a limited number of source nodes, we can use Dijkstra's shortest path algorithm to return answers in increasing order of cost. Furthermore, we can construct the product automaton incrementally.

The simpler setting in which approximation is captured by edit operations, all with cost one, can be captured by a transducer T with a single state s and transitions from s to s labelled:

- $(a, 1, \varepsilon)$, for each $a \in \Sigma$ (deletion),
- $(\varepsilon, 1, a)$, for each $a \in \Sigma$ (insertion), and
- $(a, 1, b)$, for $a, b \in \Sigma$, $a \neq b$ (substitution).

Alternatively, in [40] ideas from approximate string matching [65] can be used to construct an *approximate* NFA M_Q from Q, from which the product of M_Q and M_G can be traversed. Extending approximate matching from RPQs to CRPQs, as well as adding an inversion edit operation for reversing the traversal of a graph edge, was studied in [40, 59]. Both the approximate NFA and the product automaton can be constructed incrementally, while any rank-join algorithm can be used on the conjuncts in order to return answers in increasing overall distance from the original CRPQ. This results in an algorithm with PTIME combined complexity if the conjuncts are acyclic and there is a fixed number of head variables [40].

4. EXPRESSIVE POWER AND COMPUTATIONAL COMPLEXITY

We now consider some results on the expressive power of graph query languages and the complexity of the QEP for such languages. For simplicity of notation, let us denote the classes of queries expressible by conjunctive queries, regular path queries, conjunctive regular path queries and extended conjunctive regular path queries by CQ, RPQ, CRPQ and ECRPQ, respectively. Furthermore, let FO denote the class of queries expressible in first-order logic (relational calculus or algebra). Then we have

$$\text{CQ} \subset \text{FO}$$

and

$$\text{RPQ} \subset \text{CRPQ} \subset \text{ECRPQ}.$$

For relating these latter classes with FO and with Datalog, we need some further definitions.

The language of first-order logic with transitive closure, denoted FO + TC (introduced by Immerman in [41]) extends first-order logic with formulas of the form $TC(\lambda\bar{x}, \bar{y}.\phi(\bar{x}, \bar{y}))$, where $\bar{x}$ are $\bar{y}$ are k-tuples, and $\phi(\bar{x}, \bar{y})$ is a formula in FO+TC denoting a binary relation on k-tuples. Then $TC(\lambda\bar{x}, \bar{y}.\phi(\bar{x}, \bar{y}))$ denotes the transitive closure of ϕ.

A *linear* Datalog program is one in which each rule has at most one recursive subgoal. A *stratified* Datalog program is one in which use of negated predicates is stratified. Let SL-DATALOG and GRAPHLOG denote the sets of queries expressible as stratified linear Datalog programs and in the language GraphLog (without aggregation), respectively. Then we have the following result [18]:

$$\text{FO} + \text{TC} = \text{GRAPHLOG} = \text{SL-DATALOG}$$

Similar expressive power is exhibited by STRUQL and UnQL. A theorem from [26] states that the closure of STRUQL under composition expresses precisely the Boolean queries expressible in FO + TC, while a theorem from [13] shows that all UnCAL queries can be expressed in FO+TC (where UnCAL is the algebra associated with UnQL). Now Immerman's result tells us that all GraphLog, STRUQL and UnQL queries can be computed in NLOGSPACE.

Adding aggregation operators to GraphLog in the form of closed semirings leaves query evaluation in NLOGSPACE, as long as the summarisation operators are in $\{min, max, +\}$ and aggregation operators are in $\{min, max\}$. However, when summarisation can include $\times$ and aggregation includes $+$, as needed to express the *parts explosion* query for example, then query evaluation is in NC^2 [20]

Further study of the expressiveness of stratified aggregation in Datalog was undertaken in [53]. They show that (1) Datalog extended with stratified negation cannot express a query to count the number of paths between every pair of nodes in an acyclic graph, (2) Datalog extended with stratified negation and arithmetic on integers (the + operator) can express all computable queries on ordered domains,

and (3) Datalog extended with stratified negation and generic function symbols can express all computable queries (on ordered and unordered domains).

Recently, Fletcher et al. [27] considered the relative expressive power of navigational graph query languages built from the operators identity, union, composition, intersection, set difference, projection, co-projection (values x such that there does not exist a y such that (x, y) is in the result of some expression), converse (i.e., inverse), transitive closure, and the diversity relation (all pairs of unequal constants in the active domain). They provide a complete comparison in terms of expressive power, both for path queries and Boolean queries.

The GOOD query language provides for greater expressive power than the other languages considered above. In [33, 34], it is shown that GOOD restricted to node addition/deletion and edge addition/deletion is relationally complete. Adding abstraction gives the expressive power of the nested relational algebra, while the full language including methods is Turing-complete.

The invention of values or object identifiers (oids) also adds power, as shown in database query languages such as IQL [1] and ILOG [14]. By relying on rules that use both recursion and oid invention, it can be shown that such languages can express all computable database queries [14]. However, when recursion and invention are not allowed to interact, queries can be evaluated in PTIME [1].

5. CONCLUSION

We have provided a survey of query languages for graph databases, focussing on a number of important aspects of functionality. While many language features/constructs have been the subject of formal study, work remains to be done in terms of an integrated and consistent framework in which to study graph query languages. In particular, the areas of approximate querying and graph transformations merit greater study.

The requirements of social network modelling and analysis provide further opportunities to extend the capabilities of graph languages. For example, many aspects of social network analysis rely on some probabilistic interpretation of graphs, so query languages need to be adapted and studied accordingly. Work in the area of expressive query languages for probabilitic databases has recently been initiated [23].

Acknowledgements

I would like to dedicate this survey to the memory of Alberto Mendelzon, whose insight and vision inspired me to investigate graph query languages some 25 years ago. I would also like to thank all the colleagues with whom I have collaborated.

6. REFERENCES

[1] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. *J. ACM*, 45(5):798–842, 1998.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The LOREL query language for semistructured data. *Int. J. Digit. Libr.*, 1(1):68–88, April 1997.

[3] A. V. Aho. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 255–300. Elsevier and MIT Press, 1990.

[4] B. Amann and M. Scholl. GRAM: A graph data model and query language. In *ECHT*, pages 201–211, 1992.

[5] S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. SocialScope: Enabling information discovery on social content sites. In *CIDR*, 2009.

[6] M. K. Anand, S. Bowers, and B. Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, pages 287–298, 2010.

[7] R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[8] K. Anyanwu and A. P. Sheth. ρ-queries: enabling querying for semantic associations on the semantic web. In *WWW*, pages 690–699, 2003.

[9] M. Arenas, C. Gutiérrez, and J. Pérez. An extension of SPARQL for RDFS. In *SWDB-ODBIS*, pages 1–20, 2007.

[10] P. Barceló, C. A. Hurtado, L. Libkin, and P. T. Wood. Expressive languages for path queries over graph-structured data. In *PODS*, pages 3–14, 2010.

[11] C. L. Barrett, R. Jacob, and M. V. Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, 2000.

[12] P. Buneman. Semistructured data. In *PODS*, pages 117–121, 1997.

[13] P. Buneman, M. F. Fernandez, and D. Suciu. UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB J.*, 9(1):76–110, 2000.

[14] L. Cabibbo. The expressive power of stratified logic programs with value invention. *Inf. Comput.*, 147(1):22–56, 1998.

[15] D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.*, 64(3):443–465, 2002.

[16] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[17] M. P. Consens and A. O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *ACM Hypertext*, pages 269–292, 1989.

[18] M. P. Consens and A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS*, pages 404–416, 1990.

[19] M. P. Consens and A. O. Mendelzon. Hy$^+$: a hygraph-based query and visualization system. In *SIGMOD*, pages 511–516, 1993.

[20] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in graphlog and datalog. *Theor. Comput. Sci.*, 116(1&2):95–116, 1993.

[21] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD*, pages 323–330, May 1987.

[22] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G$^+$:

Recursive queries without recursion. In *EDS*, pages 355–368, Redwood City, 1988. Benjamin/Cummings.
[23] D. Deutch, C. Koch, and T. Milo. On probabilistic fixpoint and markov chain query languages. In *PODS*, pages 215–226, 2010.
[24] A. Dries, S. Nijssen, and L. De Raedt. A query language for analyzing networks. In *CIKM*, pages 485–494, 2009.
[25] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, pages 39–50, 2011.
[26] M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with STRUDEL. *VLDB J.*, 9(1):38–55, 2000.
[27] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. V. den Bussche, D. V. Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. In *ICDT*, pages 197–207, 2011.
[28] D. Florescu, A. Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS*, pages 139–148, 1998.
[29] D. D. Freydenberger and N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. In *AMW*, 2011.
[30] G. Grahne and A. Thomo. Regular path queries under approximate semantics. *Ann. Math. Artif. Intell.*, 46(1-2):165–190, 2006.
[31] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
[32] R. H. Güting. GraphDB: Modeling and querying graphs in databases. In *VLDB*, pages 297–308, 1994.
[33] M. Gyssens, J. Paradaens, and D. V. Gucht. A graph-oriented object database model. In *PODS*, pages 417–424, 1990.
[34] M. Gyssens, J. Paredaens, J. V. den Bussche, and D. V. Gucht. A graph-oriented object database model. *IEEE TKDE*, 6(4):572–586, August 1994.
[35] D. Harel. On visual formalisms. *C. ACM*, 31(5):514–530, May 1988.
[36] S. Harris and A. Seaborne, editors. *SPARQL 1.1 Query Language*, W3C Working Draft, 12 May 2011.
[37] H. He and A. K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. In *SIGMOD*, pages 405–418, 2008.
[38] J. Hidders. Typing graph-manipulation operations. In *ICDT*, pages 391–406, 2003.
[39] D. A. Holland, U. Braun, D. Maclean, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Choosing a data model and query language for provenance. In *Proc. Int. Provenance and Annotation Workshop*, 2008.
[40] C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In *ESWC*, pages 263–277, 2009.
[41] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.
[42] Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *PODS*, pages 40–51, 2001.
[43] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.
[44] G. Klyne and J. J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*, W3C Recommendation, 10 February 2004.
[45] K. Kochut and M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *ESWC*, pages 145–159, 2007.
[46] Z. Lacroix, H. Murthy, F. Naumann, and L. Raschid. Links and paths through life sciences data sources. In *DILS*, pages 203–211, 2004.
[47] W.-J. Lee, L. Raschid, P. Srinivasan, N. Shah, D. L. Rubin, and N. F. Noy. Using annotations from controlled vocabularies to find meaningful associations. In *DILS*, pages 247–263, 2007.
[48] U. Leser. A query language for biological networks. *Bioinformatics*, 21(2):33–39, 2005.
[49] L. Libkin and Vrgoč. Regular path queries on graphs with data. In *ICDT (to appear)*, 2012.
[50] Y. A. Liu and S. D. Stoller. Querying complex graphs. In *PADL*, pages 199–214, 2006.
[51] F. Mandreoli, R. Martoglia, G. Villani, and W. Penzo. Flexible query answering on graph-modeled data. In *EDBT*, pages 216–227, 2009.
[52] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, December 1995.
[53] I. S. Mumick and O. Shmueli. How expressive is stratified aggregation? *Ann. Math. Artif. Intell.*, 15(3-4):407–434, 1995.
[54] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB*, pages 413–424, 1996.
[55] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *ICDE*, pages 251–260, 1995.
[56] J. Paredaens, P. Peelman, and L. Tanca. G-Log: A graph-based query language. *IEEE TKDE*, 7(3):436–453, 1995.
[57] J. Pérez, M. Arenas, and C. Gutiérrez. nSPARQL: A navigational language for RDF. In *ISWC*, pages 66–81, 2008.
[58] A. Poulovassilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM TOIS*, 12(1):35–68, January 1994.
[59] A. Poulovassilis and P. T. Wood. Combining approximation and relaxation in semantic web path queries. In *ISWC*, pages 631–646, 2010.
[60] R. Ronen and O. Shmueli. SoQL: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.
[61] M. San Martín, C. Gutiérrez, and P. T. Wood. SNQL: A social networks query and transformation language. In *AMW*, 2011.
[62] A. Sheth, B. Aleman-Meza1, I. B. Arpinar, C. Bertram, Y. Warke, C. Ramakrishnan, C. Halaschek, K. Anyanwu, D. Avant, F. S. Arpinar, and K. Kochut. Semantic association identification and knowledge discovery for national security applications. *J. Database Management*, 16(1):33–53, 2005.
[63] F. W. Tompa. A data model for flexible hypertext database systems. *ACM TODS*, 7(1):85–100, January 1989.
[64] G. Weikum, G. Kasneci, M. Ramanath, and F. M. Suchanek. Database and information-retrieval methods for knowledge discovery. *C. ACM*, 52(4):56–64, 2009.
[65] S. Wu and U. Manber. Fast text searching allowing errors. *C. ACM*, 35(10):83–91, Oct. 1992.

Institute for the Management of Information Systems Athena Research Center

http://www.imis.athena-innovation.gr

1. INTRODUCTION

The Institute for the Management of Information Systems (IMIS) is part of the "Athena" Research Center, and was founded in 2007. IMIS is directed by Timos Sellis, a Professor at the School of Electrical and Computer Engineering in the National Technical University of Athens.

The mission of IMIS is to conduct research in the area of data management and large-scale information systems. This report presents an overview of its major research activities, which are currently headed by senior researchers Theodore Dalamagas, Dieter Pfoser, Yannis Stavrakas and Professor Panos Constantopoulos (Athens University of Economics and Business), and postdoctoral researchers George Papastefanatos, Dimitris Sacharidis, and Manolis Terrovitis. The institute members cooperate also with several other faculty members from universities in Greece.

2. DATA LIFECYCLES

Large data sets, and particularly curated scientific databanks, are often compared to living organisms, as they continuously evolve and introduce additional management requirements. A significant effort in IMIS is on handling the lifecycles of such evolving data collections.

Data Evolution. The key problem in reasoning about *data evolution* stems from the fact that information systems usually treat changes as distinct events. In reality, a number of changes that occur at disparate and seemingly unrelated pieces of data constitute conceptually a single "complex" change event. Such high-level changes are more meaningful than the individual changes they encompass, and offer a richer interpretation of the evolution process. In our approach, changes are discrete objects that have complex structure and retain their semantic and temporal characteristics, rather than being isolated low-level transformations on data. Consider the following simple example: the high-level change operation "move" is represented as a complex object composed by the atomic change objects "remove" and "add".

Evo-graph [37] is a novel model for data evolution that describes successive snapshots and treats changes as first-class citizens. A snapshot captures the state of the database at a specific time instance, and is represented as a rooted directed graph. A change can be compound, i.e., comprising disparate changes, and is associated with the data items it affects. Managing data evolution with evo-graphs is particularly useful when the provenance of the data needs to be traced, and past states need to be re-assessed. [38] specifies how an evo-graph can be reduced to the snapshot at a specified time instance. Furthermore, it introduces an XML representation of evo-graph, called evoXML, and presents a method to answer interesting evolution queries. A prototype, called *C2D*[1], records the history of data and the structure of changes step by step, as the current snapshot evolves [26]. Our current work is on a query language for evo-graph expressing conditions on both changes and data, and on evoXML storage optimization.

Schema Evolution. IMIS is also concerned with the management of *schema evolution* in data-centric ecosystems. Such systems, comprising a large number of applications and data stores, are highly vulnerable to schema changes. *Hecataeus* [27] is a set of tools for managing a data-centric ecosystem, i.e., the database schema along with its dependent views and queries. It satisfies the fundamental needs of the developer, administrator, and designer of a data-centric ecosystem. The developer can create evolution scenarios in order to evaluate the impact of a schema evolution event, and define rules so that both syntactical and semantic correctness is retained. The administrator can control the propagation of the event's impact to affected constructs. The designer is given an extensible suite of objective metrics that report crucial and vulnerable parts of the system regarding potential evolution events.

Digital Curation. The need to ensure adequate representation and long-term access to digital information as its context of use changes, and to counter the risk

[1] `http://web.imis.athena-innovation.gr/projects/c2d`

of repositories becoming "data mortuaries", introduces a grand challenge for digital curation research: to develop the conceptual and technological tools necessary for maintaining and adding value to a trusted body of digital information for current and future use, through the active questioning, dynamic co-evolution and adequate knowledge representation of its epistemic and pragmatic content and context.

To address this challenge, IMIS adopts (a) a *lifecycle approach* to the representation of curated information objects as these evolve in interaction with changing designated communities, (b) a *cross-disciplinary* scope, so as to cater adequately for differences in digital curation requirements between diverse scientific and functional (business, social, economic) contexts of use, (c) a broader notion of digital curation *actors*, including those involved in the production, public communication and utilization of knowledge, and (d) *event-centric* structural representations of digital information "life events".

Current lines of work include: (a) modeling digital curation processes, (b) representing domain knowledge in the form of ontologies and reference models, (c) modeling scientific and scholarly information behavior, processes and user requirements, (d) developing and maintaining knowledge resources and knowledge organization systems, (e) streamlining the enrichment of these resources from text by extracting relevant entities and relations, (f) ontology-driven search and fact discovery, (g) automatically generating text from databases as a more human-oriented form of information, to be considered for preservation purposes in addition to communication, (h) preserving contextual, schema and operational information in conjunction with primary data, so as to enable the use of databases containing valuable data over time, (i) user community modeling and social tagging, (j) conceptualizations of epistemic discourse and communication genres (i.e., rhetorical structure) in specific disciplinary and pragmatic contexts, (k) grounded research on the motives, activities and contexts of appraisal, knowledge enhancement and use of digital resources by diverse interpretive communities, and (l) cost-benefit assessment of preservation policies.

A significant part of IMIS work has contributed to a number of European projects in digital humanities infrastructures[2] and good practices[3]. Some results to date are: a digital curation lifecycle model [10, 11], an evidence-based process model of scholarly activity [3], user requirements for digital scholarly research infrastructures [6, 2], an architecture and approach to asset representation for aggregating cultural content [28], a curation-aware repository system that supports semantically-enhanced preservation services [4, 17, 18], semantic integration of collection descriptions [24], text generation from ontologies [21], ontology-based development of query patterns and their exploitation in optimizing RDF queries [12, 13].

[2] `http://www.dariah.eu` and `http://www.ehri-project.eu`

[3] `http://www.carare.eu`

3. GEOINFORMATICS

The proliferation of location-aware devices, e.g., GPS receivers, smartphones, etc., has resulted in an abundance of spatiotemporal data, which, in turn, has renewed interest in geospatial data management methods. IMIS has a long and strong interest in geoinformatics, and its research activities include indices and algorithms for road networks, as well as crowdsourcing approaches.

Road Networks. Road networks represent an interesting showcase for geospatial data management algorithms. IMIS research focuses on provision of network attributes, e.g., dynamic travel times, and the design of efficient transportation algorithms, e.g., dynamic and hierarchical routing, TSP, etc.

We introduce a hierarchical tiling scheme [16] that benefits shortest-path computation. Somewhat related, the HBA* algorithm [30] explores network hierarchies to determine the shortest path. While typically GPS technology is used to track location, our work in [5] demonstrates that wireless networks can become a complementary source especially for mobile phone users.

Optimization problems is another area of interest. In the pickup and delivery with transfers problem, the goal is to assign of a set of transportation requests to a fleet of vehicles in a way that satisfies a number of constraints and at the same time minimizes a specific cost function. [7] studies its dynamic variant and proposes a solution based on a label-setting search algorithm.

Geospatial Crowdsourcing. A research ambition of IMIS is to utilize crowdsourcing and apply smart content extraction techniques to develop large, qualitative geospatial datasets. In particular, our work [29] focuses on (a) tools for intentional provision of user-contributed spatiotemporal information (geoblogging), (b) identifying and recording geospatial observation data contained in existing content, (c) developing uncertainty as a means to represent spatial and spatiotemporal relationship data, (d) fusing data to derive an integrated geospatial datasets, and (e) developing reasoning algorithms for an evolving geospatial knowledge base.

IMIS has developed a tool [22] for collaborative search and extraction of geospatial objects from point cloud datasets such as flickr. The work in [14] summarizes the content of travel blogs by means of user sentiment, i.e., by geocoding and sentiment mapping travel blog texts, we create a heat map indicating the user sentiment (ranging from positive to negative) in relation to

geographic areas. [23] explores the annotation of travel guide content with task and location information so as to increase its usability. Narratives, in the form of travel blog entries, provide a rich resource for geospatial data, and [15] describes a method to extract geospatial route information from texts. Rather than extracting geospatial content, [31] presents a geoblogging Web-based tool that provides a simple means to author geocoded travel blog entries consisting of texts, images, video and audio.

Indexing Techniques. For a variety of problems, IMIS has utilized specialized geospatial indices. One such case is *path queries* over a large collection of geospatial routes, which is a special case of reachability queries. In particular, the goal is to construct a path between two given locations, if one exists. To expedite query processing, we introduce two complementary indexing schemes that capture transitivity information among the routes [9]. Based on these indices, our proposed algorithms are significantly faster than conventional graph traversal techniques. A more recent work [8] performs a detailed asymptotic analysis of the indices, and presents a methodology for efficiently updating them as routes are added or deleted from the collection.

For the case of distributed geospatial data, we propose a novel inherently multidimensional index, termed MIDAS [46]. MIDAS implements a distributed k-d tree, whose leaves correspond to peers, and its internal nodes dictate message routing. MIDAS requires that peers maintain little network information, and features mechanisms that support fault tolerance and load balancing. For a network of n peers, MIDAS is shown to process point and range queries in $O(\log n)$ hops in expectance.

4. WEB OF DATA

An important research area of IMIS is Web technologies, and particularly the Web of Data, described as "a Web of things in the world, described by data on the Web". IMIS has developed technologies related to Linked Open Data, RDF stores, and Web Services.

Linked Open Data. IMIS is the premium national R&D pole in Linked Open Data management and governance. The Linked Data paradigm involves practices to publish, share, and connect data on the Web, and offers a new way of data integration and interoperability. Linked Data technologies enables the Web of Data, which extends current Web to a global data space connecting data from diverse domains, and is impelled by the current trend towards an open Web. The open data movement is a significant and emerging force towards this direction. Open data is public data which are easily discoverable, accessible, and available to people without any restriction. *Linked Open Data* (LOD) serve a great cause, enabling transparency, accountability and good governance for public administrations.

IMIS has developed and launched various Open Data services. The most significant project is geodata.gov[4], a leading effort on providing services for spatial data, done in collaboration of the PM's eGov office, and the Ministry of Environment, Energy, and Climate Change. Also, IMIS has designed Linked Open Data services for microRNA databanks[5] maintained by the "A. Fleming" Biomedical Sciences Research Center (BSRC)[6], where information about biological entities are provided for the scientific community of life sciences.

The goal of IMIS is to provide innovative technologies for best governance and curation practices to produce sustainable LOD ecosystems. Our research handles the full lifecycle of LOD ecosystems, from data extraction, storage and maintenance, to monitoring, protection and repair. We focus on the following scientific and technological activities: (a) effective methods for exposing large volumes of structured and unstructured data as LOD, (b) efficient storage solutions for large volumes of LOD, (c) query methods, retrieval algorithms and ranking techniques, (d) methods for interlinking and fusing LOD from different Web data sources, (e) models and query languages to represent and query changes in LOD spaces, (f) provenance models and methods to trace the origins and transformations in LOD spaces, (g) design principles and best practices to expose LOD with anonymity guarantees, and (h) models and methods to ensure privacy for publishing LOD.

RDF Stores. In line with its LOD initiative, IMIS has developed expertise in RDF storage engines. In particular, [47] proposes a distributed storage system for RDF triples. The store is based on the distributed index of [46] and utilizes a labeling scheme to encode transitivity information within the RDF subgraphs. As a result, our store efficiently processes RDF pattern queries with known performance guarantees, and also features a forward chaining mechanism that supports RDF Schema reasoning. To exemplify its functionality, a distributed semantic content-based publish-subscribe service is implemented on top of the distributed RDF store.

Semantic Web Services. Research in IMIS also addresses the matchmaking semantic Web services, i.e., how to retrieve the best service given a user's request. If the requested service is too specific and/or the number of matching services is large, selecting the most appropriate service is a challenging task. There exist many criteria to consider for ranking Web services, such as the degree of match, quality of service, and moreover there exist many methods to quantify them. However, no sin-

[4] `http://geodata.gov.gr`
[5] `http://diwis.imis.athena-innovation.gr/mlod`
[6] `http://www.fleming.gr`

gle criterion or metric is ideal. In [35], we propose a methodology that is able to combine multiple criteria in an objective manner. To increase the utility and diversity of the returned Web services, the work in [36] extends this framework and proposes clustering method to better capture trade-offs among criteria.

5. PERSONALIZED DATA MANAGEMENT

Personalization refers to data management techniques that are centered around the user's interests, preferences, and intentions. IMIS has studied several aspects, including search and ranking, skyline queries, and indices.

Personalized Search. The general goal of personalized search is to exploit historic search data and re-rank/filter retrieved documents so as to better serve individual information needs. Traditionally, search engines have focused on improving the accuracy of results. However, accuracy does not tell the whole story. Real user experience shows that search should take into account user's extent of interest in various topics when presenting the search results. For example, this is the case when the answer set includes many documents which are similar to each other, and the resulting ranked list seems poor due to lack of diversity. A better approach is to present results in a way that covers a more diverse set of topics capturing all possible interpretations of the user's intent.

Research in IMIS includes orthogonal approaches for re-ranking search results that do not share the above limitation, and benefit all users equally [19, 20]. They are based on the observation that existing techniques focus on the retrieved content and on users search histories, but leave an important aspect unaddressed: the analysis of user search behavior. This behavior is directly observable by user feedback, by means of clicks on the results, and allows reasoning about the user's intent.

Our approach does not rely on user-specific models, but captures the user intent by grouping queries entailing similar behavior. Thus, the presented search results consist of different document types (e.g., specifications, promotion, reviews, etc.) that have been associated with products in the past. In other words, our method re-ranks the retrieved results, so that they represent the broad spectrum of user behavior for a given query. The aim is to diversify the search results in order to further enhance the exploration of the document information space. The underlying technique builds models for user intent by clustering queries with respect to the user intent, and then learns a ranking function for every cluster. The clustering and the ranking function are jointly optimized due to their dependencies.

Skyline Queries. The skyline operator has received considerable attention over the past few years as a means to retrieve tuples that are objectively good for a large set of meaningful ranking functions. The vast majority of methods are designed to efficiently process the case when every ranking attribute is totally ordered. On the other hand, our framework [34] allows the extension of almost any existing technique to handle attributes that are only partially ordered. This is quite useful for preferences specified on nominal attributes, e.g., the Make of a product, or on hierarchical attributes, e.g., an ontology. Moreover, our framework naturally extends to the dynamic case, where, for the same attribute, users may have different and conflicting preferences.

Depending on the context or underlying task, the preferences of a user may differ considerably. However, explicitly asking the user to specify her/his preferences for each context is burdensome. Instead, [32] takes a different approach. Based on the information collected so far about various contexts and user preferences, we propose a simple mechanism that uses probabilities to define preferences for the current context. Subsequently, we appropriately redefine the skyline operator so as to capture this uncertainty among preferences.

Indexing Techniques. Personalized data management can benefit from low-level performance optimizations of commonly encountered queries. One such example is boolean containment queries, motivated by the case of large set-valued tuples. Our research addresses the problem from a traditional database perspective, and focuses on data management systems that use the secondary storage to keep the indices and the data. The work in [41] proposes methods that reduce the I/O cost in basic containment queries, like subset and superset.

6. BIOLOGICAL DATA MANAGEMENT

Advances in various life sciences' applications have led to an explosive growth of experimental and computational data. IMIS establishes collaborations with biologists and designs tailor-made tools to aid their research.

Biological Analysis Tools. Biologists used to consider proteins and DNA as movers and shakers in genomics, seeing RNA as nothing more than a messenger to carry information between the two. This has changed, in early 2000s, after the discovery of the key role played in gene expression by small RNA molecules, called microRNAs. MicroRNAs can completely silence proteins by binding themselves to complementary sequences on mRNA transcripts, called *targets*. However, there is a lack of high-throughput experimental methods for identifying microRNA targets. Thus, computational methods have become increasingly important, and led to the experimental identification of many microRNA targets.

IMIS and the DNA Intelligent Analysis group (DIANA) of BSRC have designed and implemented an ad-

vanced IT infrastructure for genomic data management, oriented to processing, analysis and visualization of computationally predicted microRNA targets [25, 1]. Furthermore, IMIS and DIANA have developed a set of advanced Web applications to provide access to computationally predicted microRNA targets, including two core services: (a) DIANA microT[7] and (b) DIANA mirGen[8]. Since its original launch, these services have been one of the most widely used tool for microRNA analysis, counting more than 1,500 users per month.

Sequence Alignment. Following the collaboration between IMIS and DIANA, we have worked on a novel sequence alignment problem [48]. It has been observed that a chemical association, known as binding, of a non-coding RNA sequence, the pattern, with a gene usually occurs around a key location of the pattern, called the core (typically around the nucleotides near the start of the non-coding RNA). The Approximate Regional Sequence Matching (ARSM) problem models this chemical phenomenon and is able to predict if and where bindings occur. We have proposed an efficient algorithm that outperforms existing techniques used by biologists.

Furthermore, IMIS examines next-generation high-performance platforms for sequence alignment problems. Particularly, we employ cloud computing to efficiently detect approximate matching of pattern sequences on data sequences. This research is related to subcontracted pilot activities carried out for VENUS-C[9], an FP7 Research Infrastructures project.

7. PRIVACY PRESERVATION

Protection of privacy in information systems has attracted significant interest in the last years. By combining research and applied activities, IMIS aims at transferring its expertise to industry and academia.

Research in IMIS focuses on privacy protection methods based on data anonymization. Unlike traditional approaches, where privacy is protected by regulating access to the data, anonymization regulates the amount of information that is released in each shared dataset. The goal is to transform a dataset in a way that specific personal characteristics can no longer be discerned.

An important line of work in IMIS is privacy preservation for highly dimensional data. Examples of such data are transactional logs [40, 42, 45, 44], user movement traces [43, 39], XML files and multi-relational databases. To identify privacy threats to individuals or other entities, their respective records have to be treated as points in a multidimensional data space. The more information these records contain, the higher the dimensionality. Anonymization in this context is challenging, since adversaries can easily detect outliers, by exploiting any dimension where a record has a unique value.

Another research direction is anonymization in the presence of publicly available external knowledge. Privacy always comes at the expense of data usefulness. An interesting result of our work [33] is that it is possible to release anonymized data that are less generalized than they would be using conventional methods. We propose a generic framework that extends any existing k-anonymity algorithm so as to take into account publicly available datasets in a manner that increases the utility of the released anonymized data.

The privacy related work in IMIS is not limited to basic research, but also includes consulting services for the public and private sector and the creation of anonymization tools. IMIS is supporting the public sector in its effort to release its data to the public, by providing expertise and solutions on data anonymization.

[7] `http://diana.cslab.ece.ntua.gr/DianaTools/index.php?r=microtv4`
[8] `http://diana.cslab.ece.ntua.gr/?sec=databases`
[9] `http://www.venus-c.eu`

8. ACKNOWLEDGEMENTS

The research results presented here were made possible thanks to the dedication and hard work of the entire IMIS research team. Acknowledgments are due to Prof. Ion Androutsopoulos, Stavros Angelis, Anastasios Arvanitis, Spiros Athanasiou, Agiatis Benardou, Prof. Costis Dallas, Prof. Antonios Deligiannakis, Vicky Dritsou, Euthymios Drymonas, Alexandros Efentakis, Dimitris Gavrilis, Panos Georgantas, Giorgos Giannopoulos, Prof. Yannis Kotidis, John Liagouris, Prof. Christos Papatheodorou, Dimitrios Skoutas, and Thanasis Vergoulis.

9. REFERENCES

[1] P. Alexiou, T. Vergoulis, M. Gleditzsch, G. Prekas, T. Dalamagas, M. Megraw, I. Grosse, T. Sellis, and A. Hatzigeorgiou. mirgen 2.0: a database of microrna genomic information and regulation. *Nucleic Acids Research (Database issue)*, 38, 2010.

[2] S. Angelis, A. Benardou, P. Constantopoulos, and C. Dallas. Defining user requirements for holocaust research infrastructures and services in the ehri project. In *Proc. of the iConference*, 2012.

[3] S. Angelis, A. Benardou, P. Constantopoulos, C. Dallas, and D. Gavrilis. A conceptual model for scholarly research activity. In *Proc. of the iConference*, 2010.

[4] S. Angelis, A. Benardou, P. Constantopoulos, C. Dallas, and D. Gavrilis. A curation-aware repository supporting information-intensive scholarly research. In *Proc. of the Intl. Digital Curation Conf.*, 2010.

[5] S. Athanasiou, P. Georgantas, G. Gerakakis, and D. Pfoser. Utilizing wireless positioning as a tracking data source. In *Proc. of the Intl. Symposium on Spatial and Temporal Databases*, 2009.

[6] A. Benardou, P. Constantopoulos, C. Dallas, and D. Gavrilis. Understanding the information requirements of arts and humanities scholarship: implications for digital curation. *Intl. Journal of Digital Curation*, 5(1), 2010.

[7] P. Bouros, D. Sacharidis, T. Dalamagas, and T. Sellis. Dynamic pickup and delivery with transfers. In *Proc. of the Intl. Symposium on Spatial and Temporal Databases*, 2011.

[8] P. Bouros, D. Sacharidis, T. Dalamagas, S. Skiadopoulos, and T. Sellis. Evaluating path queries over frequently updated route collections. In *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[9] P. Bouros, S. Skiadopoulos, T. Dalamagas, D. Sacharidis, and T. Sellis. Evaluating reachability queries over path collections. In *Proc. of the Intl. Conf. on Scientific and Statistical Database Management*, 2009.

[10] P. Constantopoulos and C. Dallas. Aspects of a digital curation agenda for cultural heritage. In *Proc. of the IEEE Intl. Conf. on Distributed Human-Machine Systems*, 2008.

[11] P. Constantopoulos, C. Dallas, I. Androutsopoulos, S. Angelis, A. Deligiannakis, D. Gavrilis, Y. Kotidis, and C. Papatheodorou. DCC&U: An extended digital curation lifecycle model. *Intl. Journal of Digital Curation*, 3(1), 2009.

[12] P. Constantopoulos, V. Dritsou, and E. Foustoucos. Developing query patterns. In *Research and Advanced Technology for Digital Libraries*, 2009.

[13] V. Dritsou, P. Constantopoulos, A. Deligiannakis, and Y. Kotidis. A curation-aware repository supporting information-intensive scholarly research. In *Proc. of the Intl. Digital Curation Conf.*, 2010.

[14] E. Drymonas, A. Efentakis, and D. Pfoser. Opinion mapping travelblogs. In *Proc. of the Terra Cognita Workshop*, 2011.

[15] E. Drymonas and D. Pfoser. Geospatial route extraction from texts. In *Proc. of the ACM SIGSPATIAL Intl. Workshop on Data Mining for Geoinformatics*, 2010.

[16] A. Efentakis, D. Pfoser, and A. Voisard. Efficient data management in support of shortest-path computation. In *Proc. of the ACM SIGSPATIAL Intl. Workshop on Computational Transportation Science*, 2011.

[17] D. Gavrilis, A. Angelis, and C. Papatheodorou. Mopseus - a digital repository system with semantically enhanced preservation services. In *Proc. of the Intl. Conf. on Preservation of Digital Objects*, 2010.

[18] D. Gavrilis, C. Papatheodorou, P. Constantopoulos, and S. Angelis. Mopseus - a digital library management system focused on preservation. In *Proc. of the European Conf. on Research and Advanced Technology for Digital Libraries*, 2010.

[19] G. Giannopoulos, U. Brefeld, T. Dalamagas, and T. Sellis. Learning to rank user intent. In *Proc. of the Intl. Conf. on Information and Knowledge Management*, 2011.

[20] G. Giannopoulos, T. Dalamagas, and T. Sellis. Search behavior-driven training for result re-ranking. In *Proc. of the Intl. Conf. on Theory and Practice of Digital Libraries*, 2011.

[21] G. Karakatsiotis, D. Galanis, and I. Androutsopoulos. Naturalowl: Generating texts from owl ontologies in protege and in second life. In *Proc. of the European Conf. on Artificial Intelligence*, 2008.

[22] G. Lamprianidis and D. Pfoser. Jeocrowd: collaborative searching of user-generated point datasets. In *Proc. of the ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, 2011.

[23] J. Liagouris, S. Athanasiou, A. Efentakis, S. Pfennigschmidt, D. Pfoser, E. Tsigka, and A. Voisard. Mobile task computing: beyond location-based services and ebooks. In *Proc. of the Intl. Conf. on Web and wireless geographical information systems*, 2011.

[24] I. Lourdi, C. Papatheodorou, and M. Doerr. Semantic integration of collection description: Combining CIDOC/CRM and dublin core collections application profile. *D-Lib Magazine*, 15(7/8), 2009.

[25] M. Maragkakis, T. Vergoulis, P. Alexiou, M. Reczko, K. Plomaritou, M. Gousis, K. Kourtis, N. Koziris, T. Dalamagas, and A. Hatzigeorgiou. Diana-microt web server upgrade supports fly and worm mirna target prediction and bibliographic mirna to disease association. *Nucleic Acids Research (Webserver issue)*, 39, 2011.

[26] G. Papastefanatos, Y. Stavrakas, and T. Galani. Capturing the history and change structure of evolving data. *Submitted*.

[27] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. Hecataeus: Regulating schema evolution. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, 2010.

[28] C. Papatheodorou, C. Dallas, C. Ertmann-Christiansen, K. Fernie, D. Gavrilis, M. Masci, P. Constantopoulos, and S. Angelis. A new architecture and approach to asset representation for europeana aggregation: The CARARE way. In *Proc. of the Metadata and Semantic Research Conf.*, 2011.

[29] D. Pfoser. On user-generated geocontent. In *Proc. of the Intl. Symposium on Spatial and Temporal Databases*, 2011.

[30] D. Pfoser, A. Efentakis, A. Voisard, and C. Wenk. A new perspective on efficient and dependable vehicle routing. In *Proc. of the ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, 2009.

[31] D. Pfoser, C. Lontou, E. Drymonas, and S. Georgiou. Geoblogging: user-contributed geospatial data collection and fusion. In *Proc. of the ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, 2010.

[32] D. Sacharidis, A. Arvanitis, and T. Sellis. Probabilistic contextual skylines. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, 2010.

[33] D. Sacharidis, K. Mouratidis, and D. Papadias. k-anonymity in the presence of external databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(3), 2010.

[34] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically sorted skylines for partially ordered domains. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, 2009.

[35] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. Sellis. Top-k dominant web services under multi-criteria matching. In *Proc. of the Intl. Conf. on Extending Database Technology*, 2009.

[36] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering web services using multi-criteria dominance relationships. *IEEE Transactions on Services Computing*, 2010.

[37] Y. Stavrakas and G. Papastefanatos. Supporting complex changes in evolving interrelated web databanks. In *Proc. of the Intl. Conf. on Cooperative Information Systems*, 2010.

[38] Y. Stavrakas and G. Papastefanatos. Using structured changes for elucidating data evolution. In *Proc. of the ICDE Workshop on Managing Data Throughout its Lifecycle*, 2011.

[39] M. Terrovitis. Privacy preservation in the dissemination of location data. *ACM SIG KDD Explorations*, 13(1), 2011.

[40] M. Terrovitis. Privacy preserving publishing of set-values. *Privacy Observatory Magazine*, 2, 2011.

[41] M. Terrovitis, P. Bouros, P. Vassiliadis, T. Sellis, and N. Mamoulis. Efficient answering of set containment queries for skewed item distributions. In *Proc. of the Intl. Conf. on Extending Database Technology*, 2011.

[42] M. Terrovitis, J. Liagouris, N. Mamoulis, and S. Skiadopoulos. Privacy by disassociation. *Submitted*.

[43] M. Terrovitis and N. Mamoulis. Privacy Preservation in the Publication of Trajectories. In *Proc. of the Intl. Conf. on Mobile Data Management*, 2008.

[44] M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy-preserving Anonymization of Set-valued Data. *Proc. of the VLDB Endowment*, 1(1), 2008.

[45] M. Terrovitis, N. Mamoulis, and P. Kalnis. Local and global recoding methods for anonymizing set-valued data. *The Intl. Journal on Very Large Data Bases*, 2011.

[46] G. Tsatsanifos, D. Sacharidis, and T. Sellis. Midas: Multi-attribute indexing for distributed architecture systems. In *Proc. of the Intl. Symposium on Spatial and Temporal Databases*, 2011.

[47] G. Tsatsanifos, D. Sacharidis, and T. Sellis. On enhancing scalability for distributed rdf/s stores. In *Proc. of the Intl. Conf. on Extending Database Technology*, 2011.

[48] T. Vergoulis, T. Dalamagas, D. Sacharidis, and T. Sellis. Approximate regional sequence matching for genomic databases. *The Intl. Journal on Very Large Data Bases*, 2012.

What's new in SQL:2011

Fred Zemke, Oracle Corporation, fred.zemke@oracle.com

ABSTRACT

SQL:2011 was published in December 2011, replacing the former version (SQL:2008) as the most recent update to the SQL standard for relational databases. This paper surveys the new non-temporal features of SQL:2011.

1. INTRODUCTION

SQL (pronounced es-cue-el) is the relational database standard published jointly by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). In December 2011, ISO/IEC published the latest edition of SQL, SQL:2011.

Many readers are perhaps most familiar with SQL-86, SQL-89 and SQL-92 published by ANSI (the American National Standards Institute). Subsequently, the SQL standard has been developed internationally under the auspices of ISO/IEC. Since SQL-92, the major revisions of the SQL standard have been SQL:1999, SQL:2003, SQL:2008, and now SQL:2011. Articles in *SIGMOD Record* on prior versions of SQL may be found in references [10] - [12].

The SQL standard is published in multiple volumes, called parts. Currently there are nine parts, numbered 1, 2, 3, 4, 9, 10, 11, 13 and 14. (The gaps are for parts that have been withdrawn for various reasons; once a part number is issued, it is not recycled). Parts 3 and following were each published for the first time between the major release years (i.e., not 1992, 1999, 2003, 2008 or 2011) and subsequently "progressed" together with other extant parts. The complete list of the parts of SQL is found in the references [1] - [9].

By far the most important part is 2, Foundation [2], which is also the largest at 1470 pages (about 100 pages larger than in SQL:2008). Only five of the parts were revised in SQL:2011; for the other four parts, the version published in SQL:2008 remains in effect. This paper concentrates on the new features in SQL/Foundation:2011.

The SQL standard specifies mandatory and optional features of SQL. The mandatory features, known as Entry Level in SQL-92 and Core in the subsequent international versions, have not changed appreciably since SQL:1999. Thus the growth in the standard has been in the optional features.

Perhaps the most important new features in SQL:2011 are in the area of temporal databases. This article does not have enough space to adequately cover that topic, so it will be discussed in a future article.

The most important new non-temporal features of SQL:2011 are the following:

- DELETE in MERGE
- Pipelined DML
- CALL enhancements
- Limited fetch capability
- Collection type enhancements
- Non-enforced table constraints
- Window enhancements

Additionally, there are new DDL features to improve the usability of generated and identity columns, which are not discussed further in this article due to space limitations.

2. DELETE in MERGE

`MERGE` is a data manipulation command introduced in SQL:2003 and enhanced in SQL:2008. Here is an example that is permitted by SQL:2008. Suppose that `Inventory (Part, Qty)` is a table that lists parts and quantity on hand, and `Changes (Part, Qty, Action)` is a table of changes to be applied to `Inventory`. The `Action` column has two values, with the following meanings:

- `'Mod'` : add `Changes.Qty` to `Inventory.Qty` if the part already exists in `Inventory`.
- `'New'` : add a new row to `Inventory` using the values of `Changes.Part` and `Changes.Qty`.

In SQL:2008 this might be accomplished using the following statement:

```
MERGE INTO Inventory AS I
USING Changes AS C
ON I.Part = C.Part
WHEN MATCHED AND
  I.Action = 'Mod'
  THEN UPDATE
  SET Qty = I.Qty + C.Qty
WHEN NOT MATCHED AND
  I.Action = 'New'
  THEN INSERT (Part, Qty)
  VALUES (C.Part, C.Qty)
```

In the preceding example, the rows of `Inventory`

and Changes are matched using the join condition I.Part = C.Part. When a row of Changes matches a row of Inventory, and Action is Mod, then the row of Inventory is updated. When a row of Changes has no match in Inventory, and Action is New, then a new row is inserted in Inventory.

SQL:2011 has added the ability to perform DELETE within MERGE. This permits the following additional Action:

- 'Dis' : delete the part from Inventory since it has been discontinued.

This additional value of Action can be supported with the following command:

```
MERGE INTO Inventory AS I
USING Changes AS C
ON I.Part = C.Part
WHEN MATCHED AND
  I.Action = 'Mod'
  THEN UPDATE
  SET Qty = Qty + C.Qty
WHEN MATCHED AND
  I.Action = 'Dis'
  THEN DELETE
WHEN NOT MATCHED AND
  I.Action = 'Mod'
  THEN INSERT
  VALUES (C.Part, C.Qty)
```

The new capability in this example is the underlined DELETE, which deletes a row from Inventory.

3. Pipelined DML

Pipelined DML gives the ability to perform data change commands (INSERT, UPDATE, DELETE, MERGE) within a SELECT command.

A data change command has one or two "delta tables" containing the specific rows that are touched. A DELETE has only an old delta table (the rows to be deleted). An INSERT has only a new delta table (the rows to be inserted). An UPDATE has both an old delta table and a new delta table; the old delta table contains the "before images" and the new delta table contains the "after images". The delta table(s) of a MERGE are the unions of the old delta tables and the new delta tables of the INSERT, UPDATE and DELETE commands found within the MERGE.

Pipelined DML provides access to either the old delta table or the new delta table of a data manipulation command within a SELECT. For example,

```
SELECT Oldtable.Empno
FROM OLD TABLE (DELETE FROM Emp
                WHERE Deptno = 2)
     AS Oldtable
```

In the preceding example, the FROM clause has a DELETE command nested within it. The key words OLD TABLE indicate that rows from the old delta table of this DELETE are desired. The DELETE is executed, and afterwards, the rows of the old delta table are used to create the result, which is a list of Empno of the deleted rows.

The key words NEW TABLE may be used to access the new delta table of an INSERT, UPDATE or MERGE, for example

```
SELECT Newtable.Empno
FROM NEW TABLE (UPDATE EMP
                SET Salary = 0
                WHERE Empno > 100)
     AS Newtable
```

The preceding example sets certain salaries to 0 and returns a result consisting of the Empno of those rows whose salary is set to 0.

When using NEW TABLE, the new delta table is computed by constructing the set of new candidate rows as indicated by the INSERT, UPDATE or MERGE statement. New candidate rows may be modified by BEFORE triggers, after which they are applied to the target table. The new delta table is a snapshot of this point in query processing. There are later stages (cascaded referential actions and AFTER triggers) whose effects are not captured in the new delta table. Thus, if there are applicable cascaded referential actions or AFTER triggers, the final value of the target table may differ from the result of NEW TABLE. If the user is concerned about these, the user can specify instead FINAL TABLE. There is no "final delta table", so the FINAL TABLE option merely raises an exception if any cascaded referential action or AFTER trigger touches the target table.

4. CALL enhancements

The CALL statement, introduced in Part 4 PSM in 1996 and subsequently incorporated in Part 2 Foundation in 1999, is used to invoke an SQL-invoked procedure. Here is an example of how to create an SQL-invoked procedure using features prior to SQL:2011:

```
CREATE PROCEDURE P (
  IN A INTEGER,
  OUT B INTEGER ) ...
```

The ellipsis omits details of the syntax that specify such things as the host language of the procedure, the path to use to invoke it, etc. This example defines a procedure P with two integer parameters A and B; A is an input parameter and B is an output parameter. It is also possible to declare a parameter that is both input and output using the keyword INOUT.

Once P is defined, it can be invoked using a CALL statement, like this:

```
CALL P (1, :MyVar)
```

Here MyVar might be the name of an embedded variable that will receive the value assigned to the parameter B of P.

SQL:2011 provides two enhancements to SQL-invoked procedures:

- named arguments, and
- default input arguments.

Named arguments are illustrated in the following example:

```
CALL P (B => :MyVar, A => 1)
```

This statement is equivalent to the first example of a CALL statement. Using named arguments, the user can specify the arguments in any order.

The new default input argument feature will be illustrated using the following procedure definition:

```
CREATE PROCEDURE P (
  IN A INTEGER DEFAULT 2
  OUT B INTEGER ) ...
```

The underlined phrase DEFAULT 2 is new in SQL:2011. This syntax may be used to specify the default value of an input parameter (A in this example). Output parameters, including in-out parameters, do not support default values.

Given the preceding procedure definition, it might be invoked like this:

```
CALL P (B => :MyVar)
```

Note that this invocation does not specify the argument A. Since A is omitted, the default value 2 will be used, so the example is equivalent to

```
CALL P (B => :MYvAR, A => 2)
```

or

```
CALL P (2, :MyVar)
```

5. Limited fetch capability

A common application requirement is to fetch a subset of a query. For example, in a table of scored data, it might be desired to fetch only the top three results. Or during application development, it may be desired to fetch just ten arbitrary rows as a sample. Or in a deployed application, it may be desired to fetch only as many rows as fit in a limited display space. SQL:2008 introduced syntax to support such scenarios; this syntax has been enhanced in SQL:2011.

An example supported by SQL:2008 is

```
SELECT Name, Salary
FROM Emp
ORDER BY Salary DESCENDING
FETCH FIRST 10 ROWS ONLY
```

The preceding example will obtain the top ten wage earners from Emp. If there is a tie for the ninth, tenth and eleventh place, it is nondeterministic which two of the three ties will be returned. New in SQL:2011, one can write

```
SELECT Name, Salary
FROM Emp
ORDER BY Salary DESCENDING
FETCH FIRST 10 ROWS WITH TIES
```

The underlined key words WITH TIES (replacing the last key word ONLY in the first example) indicate that any rows tied with the tenth row should also be returned, making the result deterministic.

Another new feature is the ability to fetch a percentage of rows, as in this example:

```
SELECT Name, Salary
FROM Emp
ORDER BY Salary DESCENDING
FETCH FIRST 10 PERCENT ROWS ONLY
```

The PERCENT keyword may also be used with the WITH TIES option.

The final new feature is the ability to start the retrieval at a fixed offset, as in this example

```
SELECT Name, Salary
FROM Emp
ORDER BY Salary DESCENDING
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY
```

The underlined phrase OFFSET 10 ROWS specifies to skip the first ten rows; thus this example will retrieve the second ten highest wage earners from Emp. The underlined noise word NEXT is actually a synonym for FIRST (seen in prior examples) which the user might prefer for readability when fetching at a positive offset.

6. Collection type enhancements

SQL has two kinds of collection types: arrays (introduced in SQL:1999) and multisets (introduced in SQL:2003). Collection types are used to represent homogenous collections of elements (every element of a collection has the same data type, called the element type of the collection).

An array is an ordered collection of values, the elements of the array. The cardinality of an array is the number of elements in the array. An SQL array has variable cardinality, from zero up to a declared maximum cardinality. Thus if C is a column of array type, the cardinality of C may vary from row to row. An array may be atomically null, in which case its cardinality is regarded as null. A null array (cardinality null) is distinguished from both an empty array (cardinality 0) and an array whose every element is null (cardinality > 0).

If an array has cardinality less than the declared maximum, then the unused cells of the array are non-existent (they are not treated as implicitly null). Individual elements of an array can be referenced using square brackets to enclose a subscript; for example, `C[5]` references the fifth element of `C`.

Since SQL:1999, the cardinality of an array can be learned using the `CARDINALITY` function. New in SQL:2011, the maximum cardinality of an array can be learned using the `ARRAY_MAX_CARDINALITY` function. This is useful, for example, in writing general-purpose routines, avoiding the need to hard-code the maximum cardinality.

SQL:1999 allows assignment to a subscripted element of an array, which will either replace an existing element of the array, or increase the cardinality of the array. Any other change to an array value is done by assigning to the array as a whole, i.e., replacing the entire array. This meant that there was no way easy to remove elements from an array. New in SQL:2011, the function `TRIM_ARRAY` can be used to remove elements from the end of an array.

The other kind of collection type is multiset, which is an unordered homogenous collection that permits duplicates among the elements. There is no declared maximum cardinality for a multiset, though implementations will have physical or architectural limits. Since a multiset is unordered, there is no subscript notation to address an individual element.

New in SQL:2011, it is possible to define distinct types that are sourced from collection types. Previously, distinct types as introduced in SQL:1999 are user-defined types sourced from a predefined type. For example, a user might define `Shoesize` as a distinct type sourced from `INTEGER`. A value of `Shoesize` is an `INTEGER` value, but without the built-in semantics; for example, addition is not defined on `Shoesize`. Instead, the user can define the semantics of `Shoesize` by defining methods or other user-defined routines to manipulate values of type `Shoesize`.

In SQL:1999, the only source types for distinct types were predefined types. SQL:2011 now permits collection types as source types. For example, under SQL:1999, the user could create a column or an SQL variable that is an array of `Shoesize`, but could not create a distinct type that is array of `INTEGER`. There is no difference between an array of `Shoesize` and an array of `INTEGER` at the storage level, but semantically, SQL:1999 had no way to define methods on the array as a whole, only on the elements of the array. SQL:2011 has filled this gap by allowing distinct types sourced from collection types.

7. Non-enforced table constraints

Table constraints are declared restrictions on the possible values in rows of a table. There are three kinds: unique constraints (requiring the value in a column or set of columns to be unique across the rows of the table), referential constraints (to enforce parent-child relationships between tables) and check constraints (to enforce a Boolean condition within each row, for example, that hire date must be greater than birth date). Table constraints have been part of SQL since its inception in SQL-86.

Under most circumstances, the user wishes constraints to be enforced, since they are vital to maintaining the integrity of the data. However, there are situations in which a user wishes to temporarily turn off one or more constraint checks, such as bulk loads or replications. SQL-92 provided the ability to defer constraint checking to the end of a transaction. However, this capability does not really address the bulk load scenario, since the user will want to commit periodically during a large data transfer as a precaution against a system failure.

SQL:2011 has addressed this problem by providing syntax to alter a constraint to be either enforced or not enforced. By default, a constraint is enforced, but the user can set it to be not enforced, for example, during a bulk load. A non-enforced constraint is not checked, not even at commit time. However, when a non-enforced constraint is subsequently set to be enforced, then the constraint will be checked on all the data. Generally, such enforcement is more efficient than doing it incrementally during the load.

8. Window enhancements

Windows and window functions were first introduced via an amendment in 2000 to SQL:1999, and were incorporated directly in SQL/Foundation:2003 and subsequent editions of the standard.

To review, a window allows a user to optionally partition a data set, optionally order the rows of each partition, and finally specify a collection of rows (called the window frame) that is associated with each row. The window frame of a row *R* is some subset of the window partition of *R*. For example, the window frame may consist of all rows from the beginning of the partition up through and including *R*, according to the window ordering.

A window function is a function that computes a value for a row *R* using the collection of rows in the window frame of *R*. For example, an aggregate such as `SUM` might be computed over a window, as in this example:

```
SELECT Acctno, TransDate,
  SUM (Amount) OVER
  ( PARTITION BY Acctno
    ORDER BY TransDate
    ROWS BETWEEN
    UNBOUNDED PRECEDING
    AND CURRENT ROW )
FROM Accounts
```

In the preceding example, `Accounts` is a table containing data including `Acctno`, `TransDate` and `Amount`. The `OVER` clause specifies the window, which is partitioned by `Acctno`, ordered by `TransDate`, and finally, for each row *R*, the window frame consists of all rows from the beginning of the partition through *R*. Thus this query might be used to provide running account balances for each account number, in order of transaction date.

SQL:2011 has added the following window enhancements:

- `NTILE`
- Navigation within a window
- Nested navigation in window functions
- GROUPS option

These new features are described in the following subsections.

8.1 NTILE

`NTILE` is a window function that apportions an ordered window partition into some positive integer number *n* of buckets, numbering the buckets from 1 through *n*. If the number of rows *m* in the partition is not evenly divisible by *n*, then the extra rows are handled by making the first *r* buckets one row larger, where *r* is the remainder of the integer division *m / n*. For example,

```
SELECT Name, NTILE(3)
       OVER (ORDER BY Salary ASC)
       AS Bucket
FROM Emp
```

In this example, suppose there are 5 employees. The query asks to place them into 3 buckets. The remainder of 5/3 is 2; therefore the first two buckets will have 2 rows and the last bucket will have 1 row. Suppose the employees, in ascending order of `Salary`, are named Joe, Mary, Tom, Alice, and Frank. Then Joe and Mary are assigned to bucket 1, Tom and Alice to bucket 2, and Frank to bucket 3.

8.2 Navigation within a window

Five window functions have been added to evaluate an expression in a row *R2* at interesting places in the window frame of a current row *R1* : `LAG`, `LEAD`, `NTH_VALUE`, `FIRST_VALUE`, and `LAST_VALUE`.

8.2.1 LAG and LEAD

`LAG` and `LEAD` are window functions that provide access to a row *R2* at some offset from the current row *R1* within the window frame of *R1*. For example, given a time series of prices, suppose you wish to display a `Price` and the `Price` immediately prior in the time series. This can be done with this query:

```
SELECT Price AS CurPrice,
  LAG (Price) OVER
  (ORDER BY Tstamp) AS PrevPrice
FROM Data
```

In this example, the `Price` values in `Data` have been ordered by the timestamp `Tstamp`. For each row of `Data`, the result has two columns, `CurPrice` and `PrevPrice`. `CurPrice` is the current row's value of `Price` and `PrevPrice` is the immediately preceding value of `Price`.

The default offset, as in the preceding example, is 1 row. Other offsets may be specified as the second argument to `LAG` using an unsigned integer literal, for example

```
SELECT Price AS CurPrice,
  LAG (Price, 2) OVER
  (ORDER BY Tstamp) AS PrevPrice2
FROM Data
```

The first *n* rows, where *n* is the offset, will have no predecessor, and the `LAG` function will result in null by default. A third optional argument may be used to specify a different default, like this:

```
SELECT Price AS CurPrice,
  LAG (Price, 2, 0) OVER
  (ORDER BY Tstamp) AS PrevPrice2a
FROM Data
```

In the preceding example, in the first two rows, the value of `PrevPrice2a` is 0.

The final option on `LAG` is to compress out nulls before offsetting. This is illustrated in the following:

```
SELECT Price AS CurPrice,
  LAG (Price, 3, 0) IGNORE NULLS
  OVER (ORDER BY Tstamp)
  AS PrevPrice3
FROM Data
```

The preceding example counts backwards through three rows of non-null `Price`; if there are not that many, the value of `PrevPrice3` is 0. If desired, the keywords `RESPECT NULLS` may be used for the default behavior, which is to retain null data before counting rows backwards from the current row.

`LEAD` is essentially the same as `LAG`, except that it looks forward in the ordered window partition rather than backwards.

8.2.2 NTH_VALUE

LAG and LEAD evaluate an expression in a row *R2* that is located relative to the current row *R1*. The NTH_VALUE window function is similar, except that it navigates to a row *R2* that is at an offset from either the first or last row of the window frame of *R1*. For example,

```
SELECT Price AS CurPrice,
  NTH_VALUE (Price, 1)
  FROM FIRST
  IGNORE NULLS
  OVER ( ORDER BY Tstamp
         ROWS BETWEEN 3 PRECEDING
         AND 3 FOLLOWING )
  AS EarlierPrice
FROM Data
```

In this example, EarlierPrice is evaluated as follows:

- Form the window frame of the current row *R1*.
- Evaluate the expression Price in each row of the window frame.
- Since IGNORE NULLS is specified, remove any nulls from the collection of values.
- Starting at the first remaining value, move forward (because FROM FIRST is specified) by one row (because the offset in the second argument is 1)
- The value of EarlierPrice is the value of Price in the chosen row.

Instead of FROM FIRST, one specifies FROM LAST in order to offset from the last row of the window frame. The offset is still a positive integer, though it is used to offset backwards through the window frame.

Instead of IGNORE NULLS, one can specify RESPECT NULLS to retain nulls in the set of candidate rows prior to offsetting.

8.2.3 FIRST_VALUE and LAST_VALUE

The FIRST_VALUE and LAST_VALUE window functions are special cases of NTH_VALUE, in which the offset is always 0. FIRST_VALUE is equivalent to NTH_VALUE using the FROM FIRST option with an offset of 0, while LAST_VALUE is equivalent to NTH_VALUE using FROM LAST with an option of 0. Both FIRST_VALUE and LAST_VALUE support the choice of IGNORE NULLS or RESPECT NULLS.

8.3 Nested navigation in window functions

The window functions LAG, LEAD, NTH_VALUE, FIRST_VALUE and LAST_VALUE enable the user to evaluate an expression at a row *R2* at some point relative to the window frame of the current row *R1*. However, these functions cannot be nested within other window functions. Consider, for example, the following query: how many times in the past 30 trades has the Price been greater than the current Price? This query can be answered with a self-join, which users frequently find difficult to write, and then the DBMS finds difficult to optimize. Instead of using a self-join, it would be desirable to use a window to survey the past 30 trades. Then the problem reduces to counting the number of rows in the window frame in which the Price exceeds the current Price. Using new features in SQL:2011, the query can be expressed as follows:

```
SELECT Tstamp,
  SUM ( CASE WHEN Price >
        VALUE_OF (Price AT
                  CURRENT_ROW)
        THEN 1 ELSE 0 )
  OVER ( ORDER BY Tstamp
         ROWS BETWEEN 30 PRECEDING
         AND CURRENT ROW )
FROM Data
```

In the preceding query, the SUM is a window aggregate over a window that surveys the preceding 30 trades. The SUM is performed on a collections of 1's and 0's, so it is effectively a count of the number of 1's that are summed. There is a 1 for every Price in the window frame that exceeds the Price at the current row. To obtain the Price at the current row, the VALUE_OF function, new in SQL:2011, is used. The VALUE_OF function specifies an expression to evaluate and a row of the window frame at which to perform the evaluation. In this example, the keyword CURRENT_ROW, called a row marker, indicates the current row. Other row markers can be used to indicate the beginning or end of the window partition or the window frame. Additionally, a row marker may have an integer offset that is added or subtracted.

8.4 GROUPS option

The window frame of a row *R* consists of a set of rows in the same window partition as *R*, defined by a starting and an ending position. The starting position UNBOUNDED PRECEDING is absolute, as is the ending position UNBOUNDED FOLLOWING.. CURRENT ROW may be used as either a starting or an ending position, and is simply the position of *R*. It is also possible to specify relative starting or ending positions using an offset from *R*. For example

```
SELECT Acctno, TransDate,
  SUM (Amount) OVER
  ( PARTITION BY Acctno
    ORDER BY TransDate
    ROWS BETWEEN
    3 PRECEDING
```

```
    AND 3 FOLLOWING )
FROM Accounts
```

In the preceding example, the window frame is measures in ROWS and consists of up to 7 rows (3 before *R*, *R* itself and 3 after *R*). Alternatively, the window frame might be measured quantitatively, as in this example

```
SELECT Acctno, TransDate,
  SUM (Amount) OVER
  ( PARTITION BY Acctno
    ORDER BY TransDate
    RANGE BETWEEN
    INTERVAL '1' MONTH PRECEDING
    AND
    INTERVAL '1' MONTH FOLLOWING )
FROM Accounts
```

The preceding example uses RANGE to specify the window frame by offsetting the value of the sort column Transdate plus or minus 1 month from *R*.

The options ROWS and RANGE have their respective advantages and disadvantages. RANGE can only be used with a single sort key, and the sort key must be of a data type that supports addition and subtraction. ROWS will work with any number or data types of sort keys, but results can be non-deterministic since counting by rows may bisect a contiguous group of rows that are identical on the sort keys.

SQL:2011 introduced a third option, GROUPS, which combines some of the features of both ROWS and RANGE. GROUPS operates by counting groups of rows that are identical on the sort keys. Thus GROUPS can work with any number and data types of sort keys, and still give a deterministic result. For example,

```
SELECT Acctno, TransDate,
  SUM (Amount) OVER
  ( PARTITION BY Acctno
    ORDER BY TransDate
    GROUPS BETWEEN
    3 PRECEDING
    AND 3 FOLLOWING )
FROM Accounts
```

The window frame of a row *R* consists of up to 7 groups of rows (3 groups before *R*, the group of *R* itself, and 3 groups after *R*), where a group is a set of rows that have identical TransDate.

9. Acknowledgements

The author thanks his colleagues on the ANSI SQL committee Jim Melton (Oracle), Krishna Kulkarni (IBM) and Jan-Eike Michels (IBM) for reviewing this article.

10. References

[1] ISO/IEC 9075-1:2011, *Information technology—Database languages—SQL—Part 1: Framework (SQL/Framework)*, 2011

[2] ISO/IEC 9075-2:2011, *Information technology—Database languages—SQL—Part 2: Foundation (SQL/Foundation)*, 2011

[3] ISO/IEC 9075-3:2008, *Information technology—Database languages—SQL—Part 3: Call-Level Interface (SQL/CLI)*, 2008

[4] ISO/IEC 9075-4:2011, *Information technology—Database languages—SQL—Part 4: Persistent Stored Modules (SQL/PSM)*, 2011

[5] ISO/IEC 9075-9:2008, *Information technology—Database languages—SQL—Part 9: Management of External Data (SQL/MED)*, 2008

[6] ISO/IEC 9075-10:2008, *Information technology—Database languages—SQL—Part 10: Object Language Bindings (SQL/OLB)*, 2008

[7] ISO/IEC 9075-11:2011, *Information technology—Database languages—SQL—Part 11: Information and Definition Schemas (SQL/Schemata)*, 2011

[8] ISO/IEC 9075-13:2008, *Information technology—Database languages—SQL—Part 13: SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)*, 2008

[9] ISO/IEC 9075-14:2011, *Information technology—Database languages—SQL—Part 14: XML-Related Specifications (SQL/XML)*, 2011

[10] Andrew Eisenberg and Jim Melton, "SQL:1999, formerly known as SQL3" SIGMOD Record Vol. 28 No. 1 March 1999, zip available at http://www.sigmod.org/publications/sigmod-record/9903/index.html

[11] Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels, and Fred Zemke, "SQL:2003 has been published" SIGMOD Record Vol. 33 No. 1 March 2004 pp. 119-126 http://www.sigmod.org/publications/sigmod-record/0403/E.JimAndrew-standard.pdf

[12] Andrew Eisenberg and Jim Melton, "Advances in SQL/XML", SIGMOD Record Vol. 33 No. 3 Sept. 2004, http://www.sigmod.org/publications/sigmod-record/0409/11.JimMelton.pdf

The Curriculum Forecast for Portland: Cloudy with a Chance of Data

Michael Grossniklaus
Computer Science Department
Portland State University
Portland, OR 97201
grossniklaus@cs.pdx.edu

David Maier
Computer Science Department
Portland State University
Portland, OR 97201
maier@cs.pdx.edu

ABSTRACT

With the advent of cloud computing, new data management technologies and systems have emerged that differ from existing databases in important ways. As a consequence, universities are currently facing the challenge of integrating these topics into their curriculum in order to prepare students for the changed IT landscape. In this report, we describe the approach we have taken at Portland State University to teach data management in the cloud. We also present our experiences with this effort and give an outlook on how it could be adapted to suit the requirements of other universities.

1. MOTIVATION

Novel cloud data-management systems are different from traditional databases (and each other) in models, languages, consistency guarantees, scalability, and performance characteristics. Therefore, cloud data management has spawned activities in both the database research community and industry. In research, it has lead to renewed interest in shared-nothing architectures and alternative data processing paradigms. In industry, new and existing companies target cloud-based services and infrastructures, both as providers and users. In the Pacific Northwest and Northern California, well-known companies such as Amazon, Google, and Microsoft as well as countless startups are seeking professionals with expertise in cloud computing and cloud data management. As a consequence, we at Portland State University determined that curriculum in this area would be valuable to students entering the job market in the wider Portland area.

The first decision we had to make was whether to integrate these topics into existing undergrad and grad courses, or offer a stand-alone course on cloud data management. When redesigning their curriculum, the University of Washington, for example, opted to teach the MapReduce [4] data processing paradigm in their introductory database course at the undergrad level. In contrast to their approach, we created a dedicated 10-week course for both grad students and advanced undergrads, motivated by at least two reasons. First, due to the timely nature of this topic, we wanted to ensure that all students and, in particular, students close to their graduation get a chance to learn about cloud data management. Extending existing basic courses is therefore not an option as our target group of students will, in all likelihood, have already attended them. Second, the emerging nature of this topic does, in our opinion, not yet justify modifying the existing curriculum, which has been designed to teach established basic knowledge. We believe that a stand-alone course provides a better framework to experiment with the teaching of novel topics and that, once they have matured, blocks from such courses can be integrated into mainstream courses.

The second decision concerned the actual contents of the course. Our main goal was to make the course as self-contained and complete as possible. In terms of self-containedness, we assumed previous knowledge about the design and implementation of databases as well as programming skills, but chose to include an introductory primer on the fundamentals of cloud computing. In terms of completeness, we aimed for a good balance between general cloud data-management principles and actual cloud data-management systems that are currently in use or being developed. Given the diverse and heterogeneous world of cloud data-management systems, where new approaches emerge on a regular basis, finding this balance was challenging. To address this problem, we focused on a few representative systems in the lectures and introduced other related approaches using assignments such as readings, projects, and discussions. Additionally, we emphasized efforts to classify, compare and benchmark the various approaches and systems.

The remainder of this report is organized as follows. Section 2 introduces the structure and content

of the curriculum, while Section 3 describes assignments we created to complement the lectures. In Section 4 we report on experiences, then give an outlook on possible adaptations and future editions of the course in Section 5. Finally, resources available to fellow educators are listed in Section 6.

2. CURRICULUM

Apart from an initial cloud-computing primer and a closing look at the user's perspective, we structured the curriculum into two main parts, which respectively discussed novel NoSQL data management systems and efforts to scale traditional SQL databases.

2.1 The Basics

As mentioned, we began our course with a primer that introduced students to cloud-computing principles. In particular, we focused on utility computing in terms of pay-as-you-go models and elastic scalability as major factors that distinguish cloud computing from previous parallel or cluster-based computing paradigms. We also discussed enabling technologies such as virtualization and service-oriented architectures to provide processing power, storage, and software as commodities. This primer was in part based on UC Berkeley's view of cloud computing [2]. We concluded this introductory block by looking at the implications of cloud computing on data management in terms of providing cloud data services. The presentation of the corresponding challenges was based on the 2008 Claremont report on database research [1]. Finally, we introduced some ideas that have become household terms in cloud data management such as the CAP "theorem", eventual consistency, and BASE.

2.2 NoSQL Data Management

We structured the block on NoSQL data management into two parts, following David DeWitt's classification of the area into NoSQL OLTP and NoSQL Data Warehousing [5]. Our presentation of NoSQL OLTP data stores was based on Rick Cattell's survey [3] that distinguishes the key-value, document, and column-family data models. For our course, we also included object, graph, and array data models. For each data model, we selected a representative data store that was presented in the lecture. The chosen systems were Amazon Dynamo (key-value), MongoDB (document), BigTable (column family), Neo4j (graph), SciDB (array). To represent object databases, we invited Leon Guzenda, Chief Technology Officer at Objectivity, to talk about Objectivity/DB and InifiniteGraph.

In the context of NoSQL data warehousing, we introduced the Google File System (GFS) and MapReduce, and related them to their open-source counterparts, i.e., the Hadoop File System (HDFS) and Hadoop. We presented Pig/Pig Latin and Hive as declarative ways of specifying data processing tasks that build on Hadoop. For each of these novel data processing paradigms, we compared how it relates to traditional query processing. In particular, we talked about the challenges of executing iterative tasks or relational queries with joins. While we believe it important that students understand the motivation and advantages of new technologies, we are also convinced that they need to know about limitations in order to make informed and realistic decisions about their use in their professional life.

2.3 Scalable SQL Databases

To understand the trade-offs and techniques to horizontally scale traditional relational databases, we looked at two concrete systems. The first system introduced was VoltDB, which we used to emphasize design decisions that set it apart from conventional RDBMS servers. For example, VoltDB manages all data in main memory, maintains replicas for fault tolerance, and avoids user interactions in transactions, i.e., transactions must be registered in advance and can therefore be optimized off-line and scheduled serially.

For the second platform, we invited Michael Rys, Principal Lead Program Manager at Microsoft SQL Server, to present Microsoft SQL Azure, which supports horizontal scalability by sharding data over databases in a SQL Azure Federation, i.e., a cluster of SQL Server instances. Using this infrastructure, SQL-based MapReduce tasks can be defined and executed. Michael's talk concluded with a roadmap of how Microsoft plans to add further support for NoSQL paradigms to their data platform.

2.4 User Perspective

Towards the end of the course, we scheduled a third guest lecture given by Adam Lowry, a Portland State University graduate and Co-Founder of Urban Airship, a local startup that provides a content-based messaging platform for mobile applications. Adam's talk introduced students to the perspective of users of cloud data-management systems. In his presentation, Adam retraced the trials and tribulations of his company that lead them from problems with MongoDB to trouble with Cassandra, and experimentation with HBase. Ultimately, they moved off these platforms completely and their current infrastructure is based on PostgreSQL.

3. ASSIGNMENTS

Our course was accompanied with a series of assignments that students carried out in class or at home. The goal of these assignments was threefold. First, in-class discussion assignments gave us an opportunity to react to developments in cloud data management as they were happening, such as the release of Google Cloud SQL or VMware vFabric SQLFire. Second, homework reading assignments provided students with more details or alternative approaches. Finally, a course project exposed groups of students to practical experience with a specific cloud data-management system.

3.1 Readings

We decided to "front-load" our course in terms of reading assignments clustered primarily in the first five weeks. The first three assigned papers gave more background on topics discussed during the lectures of the corresponding week. We used this approach for the introduction of NoSQL data management by asking students to read Rick Cattell's survey [3] as well as to underpin our presentation of Amazon's Dynamo and Google's BigTable.

For each assignment, students were given a set of tasks which varied for undergrad and grad students to cater for different course requirements. A typical task would probe a student's understanding of the paper by asking him or her to contrast and compare the presented solutions with other approaches. Additionally, grad students were given more open tasks that challenged them to be creative and visionary.

3.2 Project

The setting for the course project was a system that manages and processes social network data. We selected this application scenario based on its appeal to the students, its relevance due to companies such as Facebook that are innovators in the domain of cloud data management, and its versatility, which enables a variety of use cases that range from operating a social-networking site (OLTP) to social-network analysis (OLAP). This assignment was challenging, but again we believe it is important to use a scenario that can also demonstrate some of the limitations of these new technologies.

3.2.1 Modeling

Students formed five teams with five to six students each. Each group worked with a different cloud data-management system. To span a range of models, we selected Voldemort, CouchDB, SimpleDB, Cassandra, and OrientDB. Students first familiarized themselves with their system and compiled a detailed system profile to characterize and compare it with other systems. In order to help them cover the same points, we provided a template for this task. Each group presented the resulting profile in class in order to give all students a chance to learn about the particulars of each system.

After this "warm-up" task, we specified a conceptual graph data model that teams had to express in the logical data models of their systems. We also provided a set of three example queries that the students' designs needed to support. The three queries—friends of a friend, identifying bridges, and transitive closure—represent a variety of use cases with very different complexities. Since none of the systems chosen by the students supports declarative relational queries, any data model design is closely coupled to the queries that need to be supported. The goal of this modeling exercise was making students realize that a design that works well for one type of query might hamper support for other types of queries, or even render them impossible.

3.2.2 Implementation

Students were then asked to implement their design and optionally deploy the application in the cloud. Due to the wide range of systems used, providing support for a common deployment platform was impractical. Rather, we encouraged students to investigate deployment options for their specific system. As a result, some of them used a trial version of Amazon's EC2, while the CouchDB team deployed to the Iris Couch hosting service and the SimpleDB team worked with a trial version managed by Amazon. Once again, we asked students to present their design and implementation in class following an outline we provided.

3.2.3 Report and Essay

The final project task was to write a three-page report and essay based on an outline we distributed. In the report part, students were asked to summarize their roles in the project group as well as to discuss important decisions and choices made throughout the projects. The second part was an essay on cloud-scale data-management systems in today's world of computing. Students needed to demonstrate that they can position these systems within the IT landscape, and that they are aware of their advantages and disadvantages. Finally, they were asked to conclude with their own opinions of the topic. We decided to make this task an individual assignment to give students a chance to voice their personal views and to give us the possibility of evaluating the students individually.

4. EXPERIENCES

From an instructor's perspective, our experience with this course was quite positive. Students were highly motivated and quickly caught on to the issues we intended to convey, which they demonstrated in homework assignments and discussions in class. Designing the curriculum as well as creating the lectures and assignments from scratch was challenging, but we are satisfied that our efforts have paid off.

The course was also evaluated by the department as part of the routine course evaluation and generally received favorable to good reviews. Students were also encouraged to submit comments to help us understand better what the did or did not like. From these comments, we understand that the students appreciated the current and practical nature of the course. They also liked the many different ways of learning, i.e., readings, projects, class discussions, and guest lectures. Or, as one student put it: *"Please keep this course or something like it. It was great to learn about alternative data stores."* Some students criticized the course's workload as too heavy for a non-core course or felt that some details remained vague because the discussed technologies are still very new. In summary, the feedback was a mix of positive comments as well as valid and constructive criticism.

5. OUTLOOK

We conclude by considering how future editions of this course might be adapted and how it could be enriched for universities that have semesters instead of terms. We expect that some blocks, such as the cloud-computing primer, will eventually be embedded in prerequisite courses, making room for more topics related to data management.

There are several options in terms of additional content to compensate for this refactoring or to extend the term course to a semester course. First, lectures on the database support available in commercial cloud platforms such as Windows Azure, Google's AppEngine, or VMware vFabric could be added to the course. Second, the course could go into more details on NoSQL data warehousing by introducing additional data-processing approaches such as Google's Pregel paradigm. Ideally, there would also be an exercise in this area to give students practical experience with some of these tools.

There is a trade-off in terms of the systems introduced between breadth, i.e., range and variety of systems, and depth, i.e., details, documentation, and support provided. We prioritized breadth in an effort to expose students as much as possible to the real world, where there are many systems to chose from and there are not a lot of training materials. With an average age of 28.1 years, Portland State students are more mature and experienced than the average university student, as many of them have worked or currently work. It helped to have such people on each team, who were used to installing and configuring software as well as working with bleeding-edge tools.

6. RESOURCES

The course web site[1] contains resources such as the course schedule, a comprehensive reading list, and all assignments. Fellow instructors are free to reuse any of these resources, provided that the source is acknowledged. A PowerPoint deck with more than 300 slides is also available upon request.

7. ACKNOWLEDGMENTS

The course described in this report would not have been possible without the active and committed participation of the students as well as the help and support of the administrative staff at Portland State University. We would also like to thank Keke Chen at Wright State University for sharing the slides of his cloud computing course with us. Daniel Abadi's DBMS Musings Blog[2] has been a constant source of inspiration, interesting views, and timely commentary on current events. Michael Grossniklaus' work at Portland State University is funded by the Swiss National Science Foundation (SNSF) grant number PA00P2_131452.

8. REFERENCES

[1] R. Agrawal *et al.* The Claremont Report on Database Research. *Commun. ACM*, 52:56–65, 2009.

[2] M. Armbrust *et al.* Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, University of California at Berkeley, February 2009.

[3] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Record*, 39(4):12–27, 2010.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. Symp. on Opearting Systems Design & Implementation (OSDI)*, pages 137–149, 2004.

[5] D. J. DeWitt and R. Nehme. Big Data – What's the Big Deal? `http://pages.cs.wisc.edu/~dewitt/includes/passtalks/passtalks.html`, 2011.

[1] `http://datalab.cs.pdx.edu/education/clouddbms/`
[2] `http://dbmsmusings.blogspot.com/`