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 Pauley 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>,
Michael Benedikt, Phokion G. Kolaitis, Leonid Libkin, Jan Paradaens and Thomas Schwentick.

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 September 2011 issue of the ACM SIGMOD Record. Thanks to our energetic editors, and in particular to the newcomers (Pablo Barceló for Database Principles, Vanessa Braganholo for Distinguished Profiles in Database Management, and Glenn Pauley for Industrial Perspectives), all columns are represented!
I take the opportunity to thank Leonid Libkin for his remarkable service as a Database Principles editor over twelve years. Leonid has stepped down and the column is in the good hands of Pablo now. Thank you, Leonid!

The issue starts with a Database Principles column by Calì, Gottlob, Lukasiewicz and Pieris. This article is very timely and most welcome, coming at a short interval after the remarkable related invited lecture by Georg Gottlob at ICDE 2011. Organized as a short survey, it informs our database-oriented readership on recent advancements in querying data with attached knowledge, or ontology-enhanced databases. The article outlines variants of Datalog +/- languages, and provides their complexity results.

The paper by Buneman, Cheney, Lindler and Mueller describes their Database Wiki platform for collaborative data management. The tool seeks to combine the best of both the relational database and Wiki worlds: structured, robust data management as provided by databases, and collaborative, flexible, unstructured as supported by Wikis. The platform is particularly tailored to support and simplify collaborative curation of data, by providing annotations, versioning, and provenance tracking.

This issue's Distinguished Profiles in Databases column features two interviews. Meral Özsoyoğlu is the first to discuss with Marianne her research work, life lessons and more. In her interview, Meral first discusses genealogical data management, which many readers will think they are reasonably acquainted with; in contrast, most of us know much less about metabolic pathways databases or metabolic networks, which she does a very good job at explaining here. Working across scientific disciplines is never easy, and Meral discusses the particular case of bioinformatics, pleading that the challenges and potential success deserve the risk-taking.
The second interview of the column features Divesh Srivastava. Beyond his amazingly productive career, Divesh is a great source of stories and insights. As anyone who has ever talked to him at a conference knows, he is also a seemingly endless source of new problems: after patiently listening to my VLDB demo in a freezing room at Cairo's Mena House Oberoi hotel, he playfully offered four to six other problems to work on! It was reassuring (of sorts) to learn years later that he would do the same to some of my PhD students. Read the interview to find out the role of AT&T as a communication company, what real life databases look like, how the large IIT-originating database community has formed, and more.

The report on the Data Management Group at NEC Research Labs highlights the activity of a recent team, created in 2009, and whose research work mainly focused on cloud-based data management. Under the umbrella of their CloudDB project, the authors survey their research on elastic OLTP workloads, base on the concept of transaction class which allows trading consistency for elasticity, one of the most attractive features of cloud computing. Further, the report outlines work on resource and workload management in heterogeneous clouds, and on cloud-supported mobile applications.

In the Industrial Perspectives column, Inkster, Zukowski, and Boncz present an overview of the integration of the VectorWise column store into the Ingres relational database system. In particular, the authors describe the approaches taken to the store and the server catalog, the integration of query processing, and the techniques used to convey execution plans between the systems and the return of results back to the client.

Two articles appear in this issue's Open Forum column. The one by Ameloot, Marx, Martens, Neven and Van Vees nicely summarizes the main facts, tendencies, authors and topics present in the 30 years of existence of the ACM Principles of Databases (PODS) conference. Do not miss out the lyrics of the famous song performed at the conference anniversary in Athens! Also, the authors had compiled a great set of graphs summarizing the PODS conference activity. Since all of them did not fare well in print, the paper points you to the associated Web site.
The second paper by Badia and Lemire addresses the authors' perceived insufficiency of database design methodology and techniques, at least as we have learned them and possibly teach them in school today! The authors offer a compelling set of anecdotes showing how these techniques are unused, misused and insufficient, and seek to open a conversation on these topics within the community. We found the paper controversial, but interesting food for thought.

Last but not least, the SIGMOD 2012 call for papers closes the issue: the place is Scottsdale, Arizona, and the research paper deadline is November 1st! Of course, all information can be found on the conference Web site (http://www.sigmod.org/2012/index.shtml).

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

October 2011

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)

A Logical Toolbox for Ontological Reasoning

Andrea Calì*

Department of Computer Science
and Information Systems
Birkbeck, University of London

andrea@dcs.bbk.ac.uk

Georg Gottlob*
Thomas Lukasiewicz
Andreas Pieris

Department of Computer Science
University of Oxford

name.surname@cs.ox.ac.uk

ABSTRACT

In ontology-enhanced database systems, an *ontology* on top of the extensional database expresses intensional knowledge that enhances the database schema. Queries posed to such systems are to be evaluated considering all the knowledge inferred from the data by means of the ontology; in other words, queries are to be evaluated against the logical theory constituted by the data and the ontology. In this context, tractability of query answering is a central issue, given that the data size is normally very large. This paper surveys results on a recently introduced family of Datalog-based languages, called Datalog+/-, which is a useful logical toolbox for ontology modeling and for ontology-based query answering. We present different Datalog+/- languages and related complexity results, showing that Datalog+/- can be successfully adopted due to its clarity, expressiveness and its good computational properties.

1. INTRODUCTION

Datalog (see, e.g., [1]) has been widely used as a database programming and query language for long time. It is rarely used directly as a query language in corporate application contexts. However, it is used as an inference engine for knowledge processing within several software tools, and has recently gained popularity in the context of various applications, such as web data extraction [6, 7, 25], source code querying and program analysis [28], and modeling distributed systems [2].

At the same time, Datalog has been shown to be too limited to be effectively used to model ontologies and expressive database schemata, as explained in [34]. In this respect, the main missing feature in Datalog is the possibility of expressing existential quantification in the head; this was addressed in the literature by introducing Datalog with *value invention* [10, 32].

This paper surveys recently introduced variants of Datalog, grouped in a family of languages that was named *Datalog*$^{\pm}$ (also written Datalog+/- whenever appropriate). Datalog$^{\pm}$ extends Datalog by allowing features such as existential quantifiers, the equality predicate, and the truth constant *false* to appear in rule heads. On the other hand, the resulting language has to be syntactically restricted, so as to achieve decidability, and in some relevant cases even tractability.

A basic Datalog program consists of a set of universally quantified function-free Horn clauses. The predicate symbols appearing in such a program either refer to *extensional database (EDB) predicates*, whose values are given via an input database, or to *intensional database (IDB) predicates*, whose values are computed by the program. In standard Datalog, EDB predicate symbols appear in rule-bodies only. A simple example is the program

$$\begin{aligned} s(X) &\rightarrow r(X), \\ r(X), e(X,Y) &\rightarrow r(Y), \end{aligned}$$

which takes as input EDB a directed graph, given by a binary edge relation e, plus a set of special vertices of this graph given by a unary relation s. The above program computes the set r of all vertices in the graph reachable via a directed path of nonnegative length from special vertices.

Given an EDB D and a Datalog program Σ, let us denote by $D \cup \Sigma$ the logical theory containing both the facts (i.e., ground atoms) of D and the rules of Σ. We say that a BCQ q evaluates to true over D and Σ iff $D \cup \Sigma \models q$. For Datalog$^{\pm}$ languages, the notion of query answering is the same, with the difference that rules allow for existential quantification in the head; such rules, in database parlance, are also known as *tuple-generating dependencies (TGDs)*.

For example, with TGDs we are able to express that every person has a father who, moreover, is himself a

*Alternate address: Oxford-Man Institute of Quantitative Finance, University of Oxford; E-mail: name.surname@oxford-man.ox.ac.uk

Database Principles Column. Column editors: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK. E-mail: libkin@inf.ed.ac.uk. Pablo Barceló, Department of Computer Science, University of Chile. E-mail: pbarcelo@dcc.uchile.cl.

person:

$$\begin{array}{rcl} person(X) & \rightarrow & \exists Y\, father(X,Y), \\ father(X,Y) & \rightarrow & person(Y). \end{array}$$

Note that here the relation *person*, which is supplied in the input with an initial value, is actually modified. Therefore, we no longer require (as in standard Datalog) that EDB relation symbols cannot occur in rule-heads.

Ontology querying (and possibly a number of other applications such as data exchange and web data extraction) can profit from appropriate forms of Datalog extended by the possibility of using rules with existential quantifiers in their heads. Other useful features are, for example, equality in rule-heads —rules with the equality predicate in the head are known as *equality-generating dependencies (EGDs)*, and they capture the well-known *functional dependencies* (see, e.g., [1]). Unfortunately, already for sets of TGDs alone, most basic reasoning and query answering problems are undecidable. In particular, given a database D and a set Σ of TGDs, checking whether $D \cup \Sigma \models q$ for a ground fact q is undecidable [8]. Worse than that, undecidability holds even in case both q and Σ are *fixed*, and only D is given as input [11]. It is therefore important to identify large classes of Datalog-based rule sets Σ that are expressive enough for being useful in real applications, and allow for decidable query answering. A further desirable feature is the tractability of query answering in *data complexity*, that is, the complexity calculated by considering only the data as part of the input, whereas q and Σ are fixed; this type of complexity is an important measure, because we can realistically assume that the EDB D is the only really large object in the input. The languages in the Datalog$^{\pm}$ family fulfil the above criteria.

One of the main tools used for proving favorable results about a number of Datalog$^{\pm}$ languages is the *chase procedure* [29, 31]. The chase is an algorithm that, roughly speaking, executes the rules of a Datalog$^{\pm}$ program Σ on input D in a forward chaining manner, thus inferring new knowledge by either adding new atoms or unifying symbols. In general, the chase procedure may terminate or not. The most notable syntactic restriction guaranteeing chase termination is *weak acyclicity* of TGDs, for which we refer the reader to the landmark paper [24]; more general syntactic restrictions were studied in [23, 33]. However, none of these restrictions is appropriate for ontology querying. One of the challenges of studying Datalog$^{\pm}$ is therefore to tackle the possible nonfiniteness of the chase, which complicates query answering. The first approach to query answering in case of infinite chase is found in the milestone paper by Johnson and Klug [29].

Structure of the paper. After some technical definitions in Section 2, we report in Section 3 on the class of *guarded* TGDs, where each rule body is required to have an atom that covers all body variables of the rule. We then consider the even more restricted class of *linear* TGDs, for which query answering is *first-order rewritable* (*FO-rewritable*) which means that q and Σ can be transformed into a first-order query q_Σ such that $D \models q_\Sigma$ iff $D \cup \Sigma \models q$, for every extensional database D. This property, introduced in [18] in the context of DLs, is essential if D is a very large database. It implies that query answering can be deferred to a standard query language such as (non-recursive) SQL.

We present stickiness, a completely different paradigm for tractable query answering, in Section 4. Roughly speaking, the syntactic condition that defines sticky sets of TGDs, which will be given in detail in the following of the paper, guarantees that if we perform *resolution* [1] (or more precisely a variant of it that considers existential quantification in the head; see, e.g., [15]) starting from a subgoal, the *newly* introduced variables in all obtained subgoals appear at most once in each subgoal. The stickiness condition is easily testable. Stickiness also guarantees the desirable FO-rewritability property.

In Section 5, we first deal with *negative constraints*, i.e., rules whose head is the truth constant *false* denoted by $\bot$. It turns out that negative constraints can be introduced without any increase of complexity, as query answering can be computed in the same way as without negative constraints, after the evaluation of suitable queries. We then introduce equality-generating dependencies (EGDs), which are well-known to easily lead to undecidability of query answering in their simplest form, even when combined with simple classes of TGDs such as inclusion dependencies [16, 21]. We focus on a very simple, nevertheless extremely useful class of EGDs, namely *key dependencies* (or simply *keys*). We discuss semantic and syntactic conditions ensuring that keys are usable without destroying decidability and tractability.

Section 6 briefly describes how highly relevant tractable DL languages, in particular the main languages of the well-known *DL-Lite* family [18, 35], can be modeled in the Datalog$^{\pm}$ framework. Section 7 concludes the paper, and gives some directions for further research on the topic.

2. PRELIMINARIES

In this section we recall some basics on databases, queries, tuple-generating dependencies, and the chase procedure.

General. We define the following pairwise disjoint (infinite) sets of symbols: a set Γ of *constants* (constitute the "normal" domain of a database), a set Γ_N of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as variables), and a set Γ_V of variables (used in queries and constraints). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. A lexicographic order is defined on $\Gamma \cup \Gamma_N$, such that every value of Γ_N follows all those of Γ. Sets of variables (or sequences, with a slight abuse of notation) are denoted as $\mathbf{X} = X_1, \ldots, X_k$, where $k \geqslant 0$. Let $[n]$ be the set $\{1, \ldots, n\}$, for any integer $n \geqslant 0$.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. We write r/n to denote that the predicate r has arity n. A *position* $r[i]$ (in a schema $\mathcal{R}$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r/n is a relation, and $t_1, \ldots, t_n$ are terms. Conjunctions of atoms are often identified with the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$ defined as follows: *(i)* $\varnothing$ is a substitution (the empty substitution), *(ii)* if h is a substitution, then $h \cup \{X \to Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$, and h does not already contain some $X \to Z$ with $Y \neq Z$. If $X \to Y \in h$, then we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution $h : \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$ such that: *(i)* if $t \in \Gamma$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n) \in A_1$, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n)) \in A_2$. The notion of homomorphism naturally extends to conjunctions of atoms.

Databases and Queries. A *relational instance* (or simply *instance*) I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where r/n is a predicate of $\mathcal{R}$, and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. We denote as $r(I)$ the set $\{\mathbf{t} \mid r(\mathbf{t}) \in I\}$. A *database* is a finite relational instance.

A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, has the form $p(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables of Γ_V or constants of Γ, and p is an n-ary predicate not occurring in $\mathcal{R}$. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $body(q)$. A *Boolean CQ (BCQ)* is a CQ of zero arity. The *answer* to a CQ q/n over an instance I, denoted as $q(I)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_N$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over I, denoted as $I \models q$, if $\langle\rangle \in q(I)$.

Tuple-Generating Dependencies. A *tuple-generating dependency (TGD)* σ over a schema $\mathcal{R}$ is a first-order formula $\forall\mathbf{X}\forall\mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \to \exists\mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $body(\sigma)$ and $head(\sigma)$, respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. Such σ is satisfied by an instance I for $\mathcal{R}$ if, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, then there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq I$.

We now define the notion of *query answering* under TGDs. Given a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$, i.e., $I \supseteq D$ and I satisfies Σ. The *answer* to a CQ q w.r.t. D and Σ, denoted as $ans(q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(I) \text{ for each } I \in mods(D, \Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \varnothing$. Note that query answering under (general) TGDs is undecidable [8]. This holds even when the schema and the set of TGDs are fixed [11], and also in the case of singleton sets of TGDs [4].

Following Vardi's taxonomy [39], the *data complexity* of query answering is the complexity w.r.t. the database only, while the *combined complexity* is the complexity calculated by considering also the query and the set of TGDs as part of the input.

The two problems of CQ and BCQ answering under TGDs are LOGSPACE-equivalent [20, 23, 24, 29]. Henceforth, we thus focus only on the BCQ answering problem. We also recall that query answering under TGDs is equivalent to query answering under TGDs with singleton atoms in the head [11]. This is shown by means of a transformation from general TGDs to TGDs with single-atom heads [11]. Moreover, the transformation preserves the properties of the classes of TGDs that we consider in this paper. Therefore, all results for TGDs with singleton atoms in the head carry over to TGDs with multiple head-atoms. We thus always assume w.l.o.g. (unless stated otherwise) that every TGD has a singleton atom in its head.

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [31], and later for checking query containment [29]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted instance satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*. The TGD chase rule comes in two equivalent fashions: *oblivious* and *restricted* [11], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for technical clarity. The TGD chase rule defined below is the building block of the chase.

TGD CHASE RULE: Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma : \varphi(\mathbf{X}, \mathbf{Y}) \to \exists\mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, then: *(i)* define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_N$ is a "fresh" labeled null not introduced before, and following lexicographically all those introduced so far, and *(ii)* add to D the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$, if not already in D.

Given a database D and a set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase, which leads to a (possibly infinite)

instance, denoted as $chase(D, \Sigma)$. We assume that the chase algorithm is *fair*, i.e., each TGD that must be applied during the construction of $chase(D, \Sigma)$ eventually it is applied. Due to the fairness assumption, the (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each instance $I \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to I [23, 24]. Using this fact it can be easily shown that for a BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

3. GUARDED & LINEAR DATALOG$^{\pm}$

For ontology querying purposes, we need to concentrate on cases where the chase produces an infinite universal solution, and also where no finite universal solution exists. Unfortunately, as already mentioned, query answering is undecidable in such cases. The recognition of expressive decidable classes of TGDs is a challenging problem. In this section we present the languages *guarded* and *linear Datalog*$^{\pm}$.

3.1 Guarded Datalog$^{\pm}$

We first discuss the class of *guarded* TGDs, which forms the language guarded Datalog$^{\pm}$, as a special class of TGDs which guarantees decidability of query answering, and even tractability w.r.t. data complexity. Queries relative to such TGDs can be evaluated over a finite part of the chase, whose size depends only on the query and the set of TGDs, but not on the database.

A TGD σ is *guarded* if it has a body-atom which contains all the universally quantified variables of σ. The leftmost such atom is the *guard atom* (or *guard*) of σ. The non-guard atoms are the *side atoms* of σ. For example, the TGD $r(X, Y), s(Y, X, Z) \rightarrow \exists W s(Z, X, W)$ is guarded (via the guard $s(Y, X, Z)$), while the TGD $r(X, Y), r(Y, Z) \rightarrow r(X, Z)$ is not guarded. Note that sets of guarded TGDs (with single-atom heads) are theories in the *guarded fragment* of first-order logic [3].

Complexity Results. The next theorem, presented in [11], establishes combined complexity results for conjunctive query answering under guarded Datalog$^{\pm}$. The EXPTIME and 2EXPTIME-completeness results hold even if the input database is fixed. Note that an *atomic query* is a CQ with just one body-atom.

THEOREM 1. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of guarded TGDs over $\mathcal{R}$. Let w be the maximum arity over all predicates of $\mathcal{R}$. Then, the following hold:*

a) If q is an atomic query, then deciding whether $chase(D, \Sigma) \models q$ is PTIME*-complete when Σ is fixed. The same problem is* EXPTIME*-complete if w is bounded, and* 2EXPTIME*-complete in general.*

b) If q is a non-atomic query, then deciding whether $chase(D, \Sigma) \models q$ is NP*-complete when Σ is fixed. The same problem is* EXPTIME*-complete if w is bounded, and* 2EXPTIME*-complete in general.*

The data complexity of query answering under guarded TGDs turns out to be polynomial in general, and linear in the case of atomic queries. In the sequel, let $\mathcal{R}$ be a relational schema, D be a database for $\mathcal{R}$, and Σ be a set of guarded TGDs over $\mathcal{R}$.

We first define the so-called *chase relation* for D and Σ, that is, a binary relation denoted $\xrightarrow{D,\Sigma}$, as follows. Suppose that during the construction of $chase(D, \Sigma)$ we apply a TGD $\sigma \in \Sigma$, with homomorphism h, and the atom $\underline{a}$ is obtained. Then, for each atom $\underline{b} \in body(\sigma)$, we have $h(\underline{b}) \xrightarrow{D,\Sigma} \underline{a}$. Intuitively, by exploiting $\xrightarrow{D,\Sigma}$, it is possible to extract all the chase derivations of $chase(D, \Sigma)$.

The *chase graph* for D and Σ is the directed graph with $chase(D, \Sigma)$ be the set of nodes, and having an edge from $\underline{a}$ to $\underline{b}$ if $\langle \underline{a}, \underline{b} \rangle \in \xrightarrow{D,\Sigma}$. We mark $\underline{a}$ as *guard* if $\underline{a}$ is the guard of σ. The *guarded chase forest* for D and Σ is the restriction of the chase graph for D and Σ to all atoms marked as guards and their children. The *guarded chase* of level up to $k \geqslant 0$ for D and Σ, denoted as $g\text{-}chase^k(D, \Sigma)$, is the set of all atoms in the forest of depth at most k.

It can be shown that (homomorphic images of) the query atoms are contained in a finite, initial part of the guarded chase forest, whose size is determined only by the query and the set of TGDs. However, this does not yet assure that also the whole derivation of the query atoms are contained in such a part of the guarded chase forest. This slightly stronger property is captured by the following definition.

Definition 1. We say that Σ has the *bounded guard-depth property (BGDP)* if, for each database D for $\mathcal{R}$ and for each BCQ q over $\mathcal{R}$, whenever there is a homomorphism μ that maps q into $chase(D, \Sigma)$, then there is a homomorphism λ of this kind such that all ancestors of $\lambda(body(q))$ in the chase graph for D and Σ are contained in $g\text{-}chase^k(D, \Sigma)$, where $k \geqslant 0$ depends only on q and Σ.

It is possible to show that guarded TGDs enjoy the BGDP. The proof is based on the observation that all side atoms that are necessary in the derivation of the query atoms are contained in a finite, initial portion of the guarded chase forest, whose size is determined only by the query and the set of TGDs (which is slightly larger than the one for the query atoms only). By this result, query answering under guarded TGDs is feasible in PTIME w.r.t. data complexity [11]. It is also hard for PTIME, as can be proved by reduction from propositional logic programming [12].

THEOREM 2. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set of guarded TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is* PTIME*-complete w.r.t. data complexity. If q is atomic, then*

the same problem is feasible in linear time w.r.t. data complexity.

Extensions. It is important to say that guarded TGDs can be enriched by *stratified negation*, where non-monotonic negation may be used in TGD bodies and queries. A natural stratified negation for query answering over ontologies, which is in general based on several strata of infinite models, is proposed in [12].

An expressive language, which forms a generalization of guarded Datalog$^{\pm}$, is *weakly-guarded Datalog$^{\pm}$* introduced in [11]. Roughly speaking, a set Σ of TGDs is weakly-guarded if, for each $\sigma \in \Sigma$, there exists an atom in $body(\sigma)$, called a *weak-guard*, that contains only the universally quantified variables of σ that occur at positions where a "fresh" null of Γ_N can appear during the construction of the chase (and not all the universally quantified variables).

3.2 Linear Datalog$^{\pm}$

The class of *linear* TGDs, which forms the language linear Datalog$^{\pm}$, is a variant of the class of guarded TGDs, where query answering is highly tractable w.r.t. data complexity. A TGD is *linear* if it has only one atom in its body (which is automatically a guard). Notice that linear TGDs are strictly more expressive than inclusion dependencies, which are the simplest type of TGDs with just one body-atom and one head-atom, without repetition of variables. For example, the linear TGD $supervises(X, X) \rightarrow manager(X)$, which asserts that everyone supervising her/himself is a manager, is not expressible with inclusion dependencies.

Complexity Results. Query answering under linear TGDs is PSPACE-complete w.r.t. combined complexity. This result is obtained immediately by results in [19, 26, 29, 40].

THEOREM 3. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is* PSPACE*-complete, even when the query is fixed.*

Let us now investigate the data complexity of query answering under linear TGDs. A class $\mathcal{C}$ of TGDs is *first-order rewritable* (henceforth abbreviated as *FO-rewritable*) if for every set Σ of TGDs in $\mathcal{C}$, and for every BCQ q, it is possible to construct a first-order query q_Σ such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$. Since answering first-order queries is in the highly tractable class AC_0 w.r.t. data complexity [41], it immediately follows that query answering under FO-rewritable classes of TGDs is in AC_0 w.r.t. data complexity.

We next recall the *bounded derivation-depth property*, introduced in [12], which is strictly stronger than the bounded guard-depth property. Informally, this property implies that (homomorphic images of) the query atoms along with their derivations are contained in a finite, initial part of the chase graph (rather than the guarded chase forest), whose size depends only on the query and and the set of TGDs. In the sequel, we denote by $chase^k(D, \Sigma)$ the *chase of level up to* $k \geqslant 0$ for D and Σ, that is, the set of all atoms of $chase(D, \Sigma)$ of derivation level at most k.

Definition 2. A set Σ of TGDs over a schema $\mathcal{R}$ has the *bounded derivation-depth property (BDDP)* if, for every database D for $\mathcal{R}$, and for every BCQ q over $\mathcal{R}$, whenever $chase(D, \Sigma) \models q$, then $chase^k(D, \Sigma) \models q$, where $k \geqslant 0$ depends only on q and Σ.

Clearly, in the case of linear TGDs, for every atom $\underline{a} \in chase(D, \Sigma)$, the subtree of $\underline{a}$ in the guarded chase forest is determined only by $\underline{a}$ itself. Therefore, for a single atom, its depth coincides with the number of applications of the TGD chase rule that are necessary to generate it. Therefore, the guarded chase forest coincides with the chase graph. By this observation, we obtain that linear TGDs have the bounded derivation-depth property.

It is known that if a class of TGDs enjoys the BDDP, then it is also FO-rewritable [12]. The main ideas behind the proof of this result are informally as follows. Since the derivation depth and the number of body-atoms in TGDs are bounded, the number of all database ancestors of query atoms is also bounded. Thus, the number of all non-isomorphic sets of potential database ancestors with variables as arguments is also bounded. Take the existentially quantified conjunction of every such ancestor set where the query q is answered positively. Then, the FO-rewriting of q is the disjunction of all these formulas. As an immediate consequence we get the following result.

THEOREM 4. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is in AC_0 w.r.t. data complexity.*

Small Query Rewritings. The rewriting algorithm employed in [12] in order to prove that the BDDP implies FO-rewritability is not very well-suited for practical implementations. In particular, the rewritten query obtained by applying this algorithm is of exponential size w.r.t. the given query and set of TGDs. The question that comes up is whether, under linear TGDs, a polynomially sized first-order query can be constructed. Interestingly, Gottlob and Schwentick shown recently that the answer to the above question is affirmative [27]. The key property underlying the proof of this result is the so-called *polynomial witness property* [27].

Definition 3. A class of TGDs $\mathcal{C}$ has the *polynomial witness property (PWP)* if, for every BCQ q over a

schema $\mathcal{R}$, for every database D for $\mathcal{R}$, and for every set $\Sigma \in \mathcal{C}$ of TGDs over $\mathcal{R}$, the following holds: if $chase(D, \Sigma) \models q$, then there is a sequence of at most $k \geqslant 0$ chase steps whose atoms already entail q, where k is polynomial with respect to q and Σ, and independent from D. □

As established in [27], given a BCQ q over a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$ which falls in a class that enjoys the PWP, one can compute in polynomial time a non-recursive Datalog program P of polynomial size with respect to q and Σ such that, for every database D for $\mathcal{R}$, $chase(D, \Sigma) \models q$ iff $D \models P$. Moreover, it was shown that the class of linear TGDs enjoys the PWP.

4. STICKY DATALOG$^{\pm}$

Unfortunately, none of the formalisms presented in the previous section is expressive enough to be able to express simple cases that allow for joins in rule-bodies such as the rule $r(X, Y), r(Z, X) \rightarrow s(X)$; clearly, the above rule is non-guarded since there is no body-atom that contains all the universally quantified variables. In this section, we present another Datalog$^{\pm}$ language which hinges on a paradigm, called *stickiness*, which is very different from guardedness and allows for joins in rule-bodies (with some realistic restriction to ensure decidability).

Formal Definition. The formal definition of the class of *sticky* sets of TGDs (which forms the language *sticky Datalog*$^{\pm}$) is based heavily on a variable-marking procedure called SMarking. This procedure accepts as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of Σ. Formally, SMarking(Σ) works as follows. First, we apply the so-called *initial marking* step: for each TGD $\sigma \in \Sigma$, and for each variable V in $body(\sigma)$, if there exists an atom $\underline{a}$ in $head(\sigma)$ such that V does not appear in $\underline{a}$, then we mark each occurrence of V in $body(\sigma)$. Then, we apply exhaustively (i.e., until a fixpoint is reached) the *propagation* step: for each pair of TGDs $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma$ (including the case $\sigma = \sigma'$), if a universally quantified variable V occurs in $head(\sigma)$ at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 1$, and there exists an atom $\underline{a} \in body(\sigma')$ such that at each position $\pi_1, \ldots, \pi_m$ a marked variable occurs, then we mark each occurrence of V in $body(\sigma)$. We are now ready to give the formal definition of sticky sets of TGDs.

Definition 4. Consider a set Σ of TGDs over a schema $\mathcal{R}$. Σ is *sticky* if there is no TGD $\sigma \in$ SMarking(Σ) such that a marked variable occurs in $body(\sigma)$ more than once.

Example 1. Consider the following set of TGDs. We mark the body-variables, according to the SMarking procedure, with hat, e.g., $\hat{X}$:

$$\begin{aligned} r(\hat{X}, \hat{Y}) &\rightarrow \exists Z\, r(Y, Z) \\ r(X, \hat{Y}) &\rightarrow s(X) \\ s(X), s(Y) &\rightarrow t(X, Y) \\ r(X, \hat{Y}), r(\hat{Z}, X) &\rightarrow s(X). \end{aligned}$$

The only variable that occurs more than once in the body of a TGD, i.e., the variable X in the body of the last TGD, is non-marked. Therefore, Σ is a sticky set. ■

Consider the simple database $D = \{r(a, a)\}$ and the set Σ of TGDs given in the above example. It is easy to verify that $chase(D, \Sigma)$ is infinite; recall that for ontology querying purposes we need to focus on cases where the chase is (in general) infinite. In fact, the first rule of Σ by itself leads to an infinite chase. Moreover, the third rule of Σ is a prime example of non-guardedness.

It is straightforward to see that the problem of identifying sticky sets of TGDs, that is, given a set Σ of TGDs, decide whether Σ is sticky, is feasible in polynomial time. This follows by observing that at each application of the propagation step, during the execution of the SMarking procedure, at least one body-variable is marked. Thus, after polynomially many steps the SMarking procedure terminates.

Sticky Property. It is interesting to see that the chase constructed under a sticky set of TGDs enjoys a syntactic property called *sticky property*.

Definition 5. Consider a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. Suppose that in the construction of $chase(D, \Sigma)$ we apply $\sigma \in \Sigma$, with homomorphism h, that has a variable V appearing more than once in its body, and the atoms $\underline{a}_1, \ldots, \underline{a}_k$, for $k \geqslant 1$, are generated. We say that $chase(D, \Sigma)$ has the *sticky property* if, for each atom $\underline{a} \in \{\underline{a}_1, \ldots, \underline{a}_k\}$, $h(V)$ occurs in $\underline{a}$, and also in every atom $\underline{b}$ such that $\langle \underline{a}, \underline{b} \rangle$ is in the transitive closure of $\xrightarrow{D,\Sigma}$.

Intuitively speaking, the sticky property implies that, during the construction of the chase, whenever a rule σ is applied, then the symbols (constants or nulls) which are associated (via homomorphism) to the join body-variables of σ appear in the generated atom $\underline{a}$, and also in all atoms resulting from some chase derivation involving $\underline{a}$, "sticking" to them (hence the name "sticky sets of TGDs").

As established in [14], stickiness is a sufficient condition for the sticky property of the chase.

THEOREM 5. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. If Σ is sticky, then $chase(D, \Sigma)$ enjoys the sticky property, for every database D for $\mathcal{R}$.*

Complexity Results. The next theorem, presented in [14], establishes combined complexity results for conjunctive query answering under sticky Datalog$^{\pm}$.

THEOREM 6. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a sticky set Σ of TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is* NP*-complete if Σ is fixed, and* EXPTIME*-complete in general.*

As shown in [14], the class of sticky sets of TGDs enjoys the BDDP (see Definition 2), and thus sticky sets of TGDs are FO-rewritable. The next result follows immediately.

THEOREM 7. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, deciding whether $chase(D, \Sigma) \models q$ is in* AC_0 *w.r.t. data complexity.*

Interestingly, the class of sticky sets of TGDs enjoys the PWP (see Definition 3)[13]. Therefore, sticky sets of TGDs are not only FO-rewritable, but also the constructed first-order query can be of polynomial size.

Extensions. Several convincing arguments for the usefulness of sticky sets of TGDs are given in [14]. However, sticky sets of TGDs are not expressive enough for being able to model simple cases such as the TGD $r(X, Y, X) \rightarrow \exists Z\, s(Y, Z)$; clearly, the variable X is marked, and thus the stickiness condition is violated. Note that the above rule falls in the FO-rewritable class of linear TGDs (see Subsection 3.2). A language that captures both linear and sticky Datalog$^{\pm}$, without losing the desirable property of FO-rewritability (and also the PWP), is *sticky-join Datalog$^{\pm}$* introduced in [14].

A more general class of TGDs, which is called *weakly-sticky* sets of TGDs, and which constitute *weakly-sticky Datalog$^{\pm}$*, is studied in [14]. Roughly, in a weakly-sticky set of TGDs, the variables that occur more than once in the body of a TGD are non-marked or occur at positions where a finite number of symbols can appear during the construction of the chase.

5. ADDITIONAL FEATURES

In this section we discuss how Datalog$^{\pm}$ can be extended with negative constraints and key dependencies.

5.1 Negative Constraints

A *negative constraint* (or simply *constraint*) is a first-order sentence of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\phi(\mathbf{X})$ is a conjunction of atoms (with no syntactic restrictions) and $\bot$ denotes the truth constant *false*; the universal quantifier is omitted for brevity. As we shall see in Section 6, constraints are vital when representing ontologies.

Example 2. Suppose that the unary predicates c and c' represent two classes. The fact that these two classes have no common instances can be expressed by the constraint $c(X), c'(X) \rightarrow \bot$. Moreover, if the binary predicate r represents a relationship, the fact that no instance of the class c participates to the relationship r (as the first component) can be stated by the constraint $c(X), r(X, Y) \rightarrow \bot$. ■

Checking whether a set of constraints is satisfied by a database given a set of TGDs is tantamount to query answering [12]. In particular, given a set of TGDs Σ_T, a set of constraints $\Sigma_\bot$, and a database D, for each constraint $\nu : \phi(\mathbf{X}) \rightarrow \bot$ we evaluate the BCQ $p \leftarrow \phi(\mathbf{X})$ over $D \cup \Sigma_T$. If at least one of such queries answers positively, then $D \cup \Sigma_T \cup \Sigma_\bot \models \bot$ (i.e., the theory is inconsistent), and thus $D \cup \Sigma_T \cup \Sigma_\bot$ entails every BCQ; otherwise, given a BCQ q, we have that $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff $D \cup \Sigma_T \models q$ (or equivalently $chase(D, \Sigma) \models q$), i.e., we can answer q by ignoring the constraints.

THEOREM 8. *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, a set Σ_T of TGDs over $\mathcal{R}$, and a set $\Sigma_\bot$ of constraints over $\mathcal{R}$. Then, $D \cup \Sigma_T \cup \Sigma_\bot \models q$ iff (i) $chase(D, \Sigma_T) \models q$ or (ii) $chase(D, \Sigma_T) \models q_\nu$, for some constraint $\nu \in \Sigma_\bot$.*

As an immediate consequence, constraints do not increase the complexity of BCQ answering under TGDs alone [12].

5.2 Key Dependencies

The addition of key dependencies (KDs) [1] is more problematic than that of constraints, since the former easily leads to undecidability of query answering (see, e.g., [16]). For this reason, the restricted class of *non-conflicting KDs*, which has a controlled interaction with TGDs, and thus decidability of query answering is guaranteed, was proposed in [12]. Nonetheless, as we shall see in Section 6, this class is expressive enough for modeling ontologies.

A *key dependency (KD)* κ is an assertion of the form $key(r) = \mathbf{A}$, where r is a predicate symbol and $\mathbf{A}$ is a set of attributes of r. It is equivalent to the set of EGDs $\{r(\mathbf{X}, Y_1, \ldots, Y_m), r(\mathbf{X}, Y'_1, \ldots, Y'_m) \rightarrow Y_i = Y'_i\}_{i \in [m]}$, where the variables $\mathbf{X} = X_1, \ldots, X_n$ appear exactly at the attributes of $\mathbf{A}$ (w.l.o.g., the first n attributes of r). Such a KD κ is *applicable* to a set of atoms B iff there exist two (distinct) tuples $\mathbf{t_1}, \mathbf{t_2} \in \{\mathbf{t} \mid r(\mathbf{t}) \in B\}$ such that $\mathbf{t_1}[\mathbf{A}] = \mathbf{t_2}[\mathbf{A}]$, where $\mathbf{t}[\mathbf{A}]$ is the projection of tuple $\mathbf{t}$ over $\mathbf{A}$. If there exists an attribute $i \notin \mathbf{A}$ of r such that $\mathbf{t_1}[i]$ and $\mathbf{t_2}[i]$ are two (distinct) constants of Γ, then there is a *hard violation* of κ, and the chase *fails*. Otherwise, the result of the application of κ to B is the set of tuples obtained by either replacing each occurrence of $\mathbf{t_1}[i]$ in B with $\mathbf{t_2}[i]$, if $\mathbf{t_1}[i]$ follows lexicographically $\mathbf{t_2}[i]$, or vice-versa otherwise.

The chase of a database D, in the presence of two sets Σ_T and Σ_K of TGDs and KDs, respectively, is computed by iteratively applying: *(i)* a single TGD once, and *(ii)* the KDs as long as they are applicable.

We continue by introducing the semantic notion of separability [16, 12], which formulates a controlled interaction of TGDs and KDs, so that the KDs do not increase the complexity of query answering.

Definition 6. Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ over $\mathcal{R}$, where Σ_T and Σ_K are sets of TGDs and KDs, respectively. Then, Σ is *separable* iff for every database D for $\mathcal{R}$ the following conditions are satisfied: *(i)* if $chase(D, \Sigma)$ fails, then there is a hard violation of some KD $\kappa \in \Sigma_K$, when κ is applied directly on D, and *(ii)* if there is no chase failure, then for every BCQ q over $\mathcal{R}$, $chase(D, \Sigma) \models q$ iff $chase(D, \Sigma_T) \models q$.

In the presence of separable sets of TGDs and KDs, the complexity of query answering is the same as in the presence of the TGDs alone. This was established in [12] (generalizing [16]) by showing that in such a case we can first perform a preliminary check to see whether the chase fails, which has the same complexity as BCQ answering, and if the chase does not fail, then proceed with query answering under the TGDs alone.

We now give the formal definition of the class of non-conflicting KDs, as defined in [12], which is actually a sufficient syntactic condition for separability. Let us say that the class of non-conflicting KDs generalizes the class of *non-key-conflicting inclusion dependencies* introduced in [16]. This condition is crucial for using TGDs to capture ontology languages, as we shall see in Section 6. Notice that, in the following definition, TGDs are assumed w.l.o.g. to have single-atom heads.

Definition 7. Let $\mathcal{R}$ be a relational schema. Consider a TGD $\sigma : \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$, and a set Σ_K of KDs over $\mathcal{R}$. We say that Σ_K is *non-conflicting (NC)* relative to σ if for each $\kappa \in \Sigma_K$ of the form $key(r) = \mathbf{A}$, the following conditions are satisfied: *(i)* the set of the attributes of r in $head(\sigma)$ where a universally quantified variable occurs is not a strict superset of $\mathbf{A}$, and *(ii)* each existentially quantified variable in σ occurs just once. We say that Σ_K is NC relative to a set Σ_T of TGDs if Σ_K is NC relative to each TGD $\sigma \in \Sigma_T$.

Example 3. Consider the TGD σ of the form $p(X, Y) \rightarrow \exists Z\, r(X, Y, Z)$, and the KDs $\kappa_1 : key(r) = \{1, 2\}$ and $\kappa_2 : key(r) = \{1\}$. Clearly, the set of the $\forall$-attributes of r in $head(\sigma)$ is $\mathbf{U} = \{1, 2\}$. Observe that $\{\kappa_1\}$ is NC relative to σ; roughly, every atom generated during the chase by applying σ will have a "fresh" null of Γ_N in some key attribute of κ_1, thus never firing this KD. On the contrary, $\{\kappa_2\}$ is not NC relative to σ since $\mathbf{U} \supset \{1\}$. ∎

6. ONTOLOGY QUERYING

In this section we briefly describe how the main languages of the well-known DL-Lite family of DLs [18, 35], namely, DL-Lite$_{\mathcal{F}}$, DL-Lite$_{\mathcal{R}}$ and DL-Lite$_{\mathcal{A}}$ can all of them be reduced to linear (resp., sticky) Datalog$^{\pm}$ with (negative) constraints and non-conflicting KDs, called *linear* (resp., *sticky*) *Datalog*$^{\pm}[\bot, =]$, and that the former are strictly less expressive than the latter. Let us recall that DL-Lite$_{\mathcal{R}}$ is able to fully capture the (DL fragment of) RDF Schema [9], the vocabulary description language for RDF; see [22] for a translation.

Intuitively, DLs model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations on classes of individuals, respectively. For instance, a DL knowledge base (or ontology) in DL-Lite$_{\mathcal{F}}$ encodes subset relationships between concepts and between roles, the membership of individuals to concepts and of pairs of individuals to roles, and functional dependencies on roles. The following example, taken from [12], illustrates some DL axioms in DL-Lite$_{\mathcal{F}}$ and their translation into Datalog$^{\pm}$ rules.

Example 4. The following are some concept inclusion axioms, which informally express that *(i)* conference and journal papers are articles, *(ii)* conference papers are not journal papers, *(iii)* every scientist has a publication, *(iv) isAuthorOf* relates scientists and articles:

$$\begin{aligned} \mathit{ConPaper} &\sqsubseteq \mathit{Article}, \\ \mathit{JouPaper} &\sqsubseteq \mathit{Article}, \\ \mathit{ConPaper} &\sqsubseteq \neg \mathit{JouPaper}, \\ \mathit{Scientist} &\sqsubseteq \exists \mathit{isAuthorOf}, \\ \exists \mathit{isAuthorOf} &\sqsubseteq \mathit{Scientist}, \\ \exists \mathit{isAuthorOf}^{-} &\sqsubseteq \mathit{Article}. \end{aligned}$$

They are translated into the following TGDs and constraints (we identify atomic concepts and roles with their predicates):

$$\begin{aligned} \mathit{ConPaper}(X) &\rightarrow \mathit{Article}(X), \\ \mathit{JouPaper}(X) &\rightarrow \mathit{Article}(X), \\ \mathit{ConPaper}(X), \mathit{JouPaper}(X) &\rightarrow \bot, \\ \mathit{Scientist}(X) &\rightarrow \exists Z\, \mathit{isAuthorOf}(X, Z), \\ \mathit{isAuthorOf}(X, Y) &\rightarrow \mathit{Scientist}(X), \\ \mathit{isAuthorOf}(Y, X) &\rightarrow \mathit{Article}(X). \end{aligned}$$

The following role inclusion and functionality axioms express that *(v) isAuthorOf* is the inverse of *hasAuthor*, and *(vi) hasFirstAuthor* is a functional binary relationship:

$$\begin{aligned} \mathit{isAuthorOf}^{-} &\sqsubseteq \mathit{hasAuthor}, \\ \mathit{hasAuthor}^{-} &\sqsubseteq \mathit{isAuthorOf}, \\ (\mathsf{funct}\ \mathit{hasFirstAuthor}). & \end{aligned}$$

They are translated into the following TGDs and KDs:

$$\begin{aligned} &\mathit{isAuthorOf}(Y, X) \rightarrow \mathit{hasAuthor}(X, Y), \\ &\mathit{hasAuthor}(Y, X) \rightarrow \mathit{isAuthorOf}(X, Y), \\ &\mathit{hasFirstAuthor}(X, Y), \mathit{hasFirstAuthor}(X, Y') \rightarrow Y = Y'. \end{aligned}$$

The following concept and role memberships express that the individual i_1 is a scientist who authors the ar-

ticle i_2:

$$Scientist(i_1),\ isAuthorOf(i_1, i_2),\ Article(i_2).$$

They are translated to identical database atoms (where we also identify individuals with their constants). ■

Formally speaking, every knowledge base $\mathcal{K}$ in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, is translated into a database $D_\mathcal{K}$, a set of TGDs Σ_T, a set of KDs Σ_K, and a set of constraints $\Sigma_\perp$. Notice that Σ_T is a set of linear TGDs and also a sticky set of TGDs. Moreover, Σ_K is non-conflicting relative to Σ_T. The next result, established in [12, 14], shows that query answering under DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, knowledge bases can be reduced to query answering under linear and sticky Datalog$^\pm[\perp, =]$.

THEOREM 9. *Let $\mathcal{K}$ be a knowledge base in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$, and q be a BCQ for $\mathcal{K}$. If $D_\mathcal{K} \models \Sigma_K$, then q holds in $\mathcal{K}$ iff either (i) chase$(D_\mathcal{K}, \Sigma_T) \models q$, or (ii) chase$(D_\mathcal{K}, \Sigma_T) \models q_\nu$, for some constraint $\nu \in \Sigma_\perp$.*

The next result follows immediately from the fact that the simple linear and sticky TGD $r(X) \rightarrow s(X, X)$ is not expressible in DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$ [12].

THEOREM 10. *Both linear and sticky Datalog$^\pm[\perp, =]$ are strictly more expressive than DL-Lite$_X$, where $X \in \{\mathcal{F}, \mathcal{R}, \mathcal{A}\}$.*

By observing that concept products [38], that is, rules of the form $p(X), q(Y) \rightarrow r(X, Y)$ which express the cartesian product of two concepts (unary relations) p and q, are very special cases of sticky sets of TGDs, it is possible to show that the above DL-Lite languages can be extended with concept product, without increasing the complexity of query answering.

7. DISCUSSION AND FUTURE WORK

In this paper we have surveyed some of the key languages in the Datalog$^\pm$ famliy. From a database point of view, these languages are in fact syntactically defined sets of TGDs (possibly enriched with non-monotonic negation and other features) that are especially suited for ontological query answering. We believe that the Datalog$^\pm$ family is a useful logical toolbox for tackling ontology reasoning tasks. Datalog$^\pm$ languages have a simple syntax and are easy to understand; they are decidable and enjoy good complexity properties. They can easily express very popular DL languages, and in addition they can be extended by non-monotonic stratified negation, a desirable feature which is not expressible in current DL languages.

Datalog$^\pm$ is still the subject of active research, and there are many challenging research problems to be tackled, some of wich we list below.

- We would like to find more general decidable fragments; the first goal is to combine the two tractability paradigms guardedness and stickiness in a natural way.
- *Transitive closure*, introduced in some expressive DL languages in a limited form, is easily expressible in Datalog, but only through non-guarded rules, whose addition to decidable sets of rules may easily lead to undecidability. We would like to study whethere there are limited forms of transitive closure that can be safely added to various versions of Datalog$^\pm$.
- A class of TGDs is *finite controllable* if it guarantees that query answering under arbitrary (finite or infinite) models coincide with query answering under finite models only. Finite controllability was shown recently for the guarded fragment of first-order logic [5], and thus holds for guarded TGDs (and it easily extends to weakly-guarded TGDs). We plan to study this property in the context of sticky sets of TGDs.
- For non-finitely-controllable Datalog$^\pm$ languages, we would like to study the complexity of query answering under finite models. Pioneering work on finite model reasoning was done in [17, 30, 36, 37].
- We plan to study optimizations of rewritings obtained for FO-rewritable Datalog$^\pm$ languages. As for now, such rewritings are in general quite large and therefore not very efficient in practice.

Acknowledgments. This work was supported by the European Research Council under the European Community's 7-th Framework Programme (FP7/2007-2013)/ERC grant no. 246858 – DIADEM.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] P. Alvaro, W. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. C. Sears. Towards scalable architectures for clickstream data warehousing. Technical report, EECS Department, University of California, Berkeley, 2009.

[3] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27:217–274, 1998.

[4] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[5] V. Bárány, G. Gottlob, and M. Otto. Querying the guarded fragment. In *Proc. of LICS*, pages 1–10, 2010.

[6] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Proc. of VLDB*, pages 119–128, 2001.

[7] R. Baumgartner, W. Gatterbauer, and G. Gottlob. Monadic Datalog and the expressive

power of web information extraction languages. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 3465–3471. Springer-Verlag New York, Inc., 2009.

[8] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP*, pages 73–85, 1981.

[9] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, 2004. W3C Recommendation.

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

[11] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.

[12] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, pages 77–86, 2009.

[13] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology langugaes: The query answering problem. Unpublished Manuscript.

[14] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[15] A. Calì, G. Gottlob, and A. Pieris. Query rewriting under non-guarded rules. In *Proc. AMW*, 2010.

[16] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS*, pages 260–271, 2003.

[17] D. Calvanese. Finite model reasoning in description logics. In *Proc. of KR*, pages 292–303, 1996.

[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[19] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. Comput. Syst. Sci.*, 28:29–59, 1984.

[20] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOCS*, pages 77–90, 1977.

[21] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies. *SIAM Journal of Computing*, 14:671–677, 1985.

[22] J. de Bruijn and S. Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *Proc. of ISWC*, pages 86–99, 2007.

[23] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisisted. In *Proc. of PODS*, pages 149–158, 2008.

[24] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[25] G. Gottlob and C. Koch. Monadic Datalog and the expressive power of web information extraction languages. *J. ACM*, 51(1):71–113, 2004.

[26] G. Gottlob and C. H. Papadimitriou. On the complexity of single-rule Datalog queries. *Inf. and Comput.*, 183(1):104–122, 2003.

[27] G. Gottlob and T. Schwentick. Rewriting ontological queries into small non-recursive Datalog programs. In *Proc. of DL*, 2011.

[28] E. Hajiyev, M. Verbaere, and O. de Moor. *codeQuest*: scalable source code queries with Datalog. In *Proc. of ECOOP*, pages 2–27, 2006.

[29] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[30] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. *Inf. Comput.*, 199(1-2):132–171, 2005.

[31] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[32] D. Mailharrow. A classification and constraint-based framework for configuration. *Artif. Intell. for Engineering Design, Analysis and Manufacturing*, 12(4):383–397, 1998.

[33] B. Marnette. Generalized schema-mappings: from termination to tractability. In *Proc. of PODS*, pages 13–22, 2009.

[34] P. F. Patel-Schneider and I. Horrocks. A comparison of two modelling paradigms in the semantic web. *J. Web Semantics*, 5(4):240–250, 2007.

[35] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[36] R. Rosati. Finite model reasoning in DL-lite. In *Semantic Web Conf.*, pages 215–229, 2008.

[37] R. Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.*, 77(3):572–594, 2011.

[38] S. Rudolph, M. Krötzsch, and P. Hitzler. All elephants are bigger than all mice. In *Description Logics*, 2008.

[39] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC*, pages 137–146, 1982.

[40] M. Y. Vardi, 1984. Personal communication reported in [29].

[41] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS*, pages 266–276, 1995.

The Database Wiki Project: A General-Purpose Platform for Data Curation and Collaboration

Peter Buneman, James Cheney, Sam Lindley
School of Informatics
University of Edinburgh
Edinburgh, United Kingdom
{opb, jcheney}@inf.ed.ac.uk, Sam.Lindley@ed.ac.uk

Heiko Mueller
Tasmanian ICT Centre
CSIRO
Hobart, Australia
heiko.mueller@csiro.au

ABSTRACT

Databases and wikis have complementary strengths and weaknesses for use in collaborative data management and data curation. Relational databases, for example, offer advantages such as scalability, query optimization and concurrency control, but are not easy to use and lack other features needed for collaboration. Wikis have proved enormously successful as a means to collaborate because they are easy to use, encourage sharing, and provide built-in support for archiving, history-tracking and annotation. However, wikis lack support for structured data, efficiently querying data at scale, and localized provenance and annotation. To achieve the best of both worlds, we are developing a general-purpose platform for collaborative data management, called DBWIKI. Our system not only facilitates the collaborative creation of structured data; it also provides features not usually provided by database technology such as annotation, citability, versioning, and provenance tracking. This paper describes the technical details behind DBWIKI that make it easy to create, correct, discuss, and query structured data, placing more power in the hands of users while managing tedious details of data curation automatically.

1. INTRODUCTION

Curated databases are finding use in all branches of science and scholarship. Most curated databases are created and maintained in a collaborative effort by a dedicated group of people – the curators – who produce a definitive reference work for some subject area. Common examples include UniProt, a resource of protein sequence and functional information [8], and IUPHAR-DB, the official database of the IUPHAR Committee on Receptor Nomenclature and Drug Classification [7], which contains contributions of a large community of experts in the field. Some curated databases are also being developed in support of "citizen science", where the public at large can contribute to the database (see [3] for examples). A system that maintains curated databases faces several technical and usability challenges [20, 13]:

1. Past versions of data need to be archived and easy to retrieve. The archiving system should also support temporal queries over the history of data.
2. Much curated data is copied and edited from existing sources. Since the value of curated databases lies in their quality and organization, knowing the origin of the curated data — its provenance — is particularly important.
3. In addition to the actual data, curated databases carry additional valuable annotations consisting of opinions of curators about the quality of data or suggested changes.
4. Curators should receive credit for their contributions. Thus, the system needs to make data items citable and attributable to their contributors.
5. Curated databases are collections of entries that predominantly follow a common structure. This common structure (or database schema) may need to change over time as the subject area evolves.
6. Many data curation projects rely on their web presence to distinguish themselves from other projects; in fact, journals such as *Nucleic Acids Research* require databases to maintain Web interfaces in order to be considered for publication.

It can be seen from these requirements that we are asking for a mixture of functionalities provided by databases and wikis. The ability to handle structured data is to some extent already present in wikis through the use of infoboxes, but they fall far short of the functionality of a database; in fact they are not even intended as the primary representation of the data they contain [6]. Relational databases, on the other hand, provide little in the way of generic support for these features. However, while it is always possible to add them for individual applications, the ability to do this largely remains the preserve of professional programmers and database admin-

istrators and requires substantial coding effort. The expense of doing this is unrealistic for many small projects that lack the resources to employ this expertise.

We believe that the needs of database curation projects could be met more reliably and cost-effectively by developing new general-purpose systems that combine the advantages of databases and wikis. We call such systems *Database Wikis*. Much of the basic research on curated databases needed to implement database wikis, such as archiving, citation, provenance, and annotation management, has already been conducted [10, 11, 13, 21]. However, there as yet is no single system that draws these techniques together.

1.1 Contributions

We are developing DBWIKI, a Database Wiki that aims to combine the ease of use and flexibility of a wiki with the robustness and scalability of a database; furthermore, DBWIKI provides unified generic techniques for database curation that have previously been prototyped in separate systems.

DBWIKI provides the ability to create, populate and browse curated databases using a standard web browser. Data entry and modification is done either using system-generated web forms or by import from other data sources such as XML files. Each piece of information has a provenance record. All changes to the data and the database schema are logged in the database. The provenance and prior versions of each individual data item are browsable through the user interface. Moreover, each piece of information within the database can be annotated (including annotations themselves). Annotations are free-form text that allow curators to share and discuss their opinions, much like comments on blogs or forums; however, annotations can be attached to any part of the data, not just pages. To demonstrate the full capabilities of DBWIKI we have used it on data from several existing curated databases, including the CIA World Factbook [2], DBLP [4], and IUPHAR-DB [7].

In addition to the database capabilities, DBWIKI includes a declarative "markdown" language for defining wiki pages. We extended a markdown syntax parser with syntax for embedded queries for viewing data. This extension enables querying and aggregating information from the curated database, *e.g., list the population of countries in Europe*. Similar proposals have been made for embedding SPARQL queries in other wiki extension projects such as Semantic MediaWiki [1]. DBWIKI's query capabilities, however, go beyond querying the current state of the data, as we also allow embedded queries over the history and the provenance of data, *e.g., list all changes made to the population of Greece*. Such queries are not currently supported by other wiki extension projects.

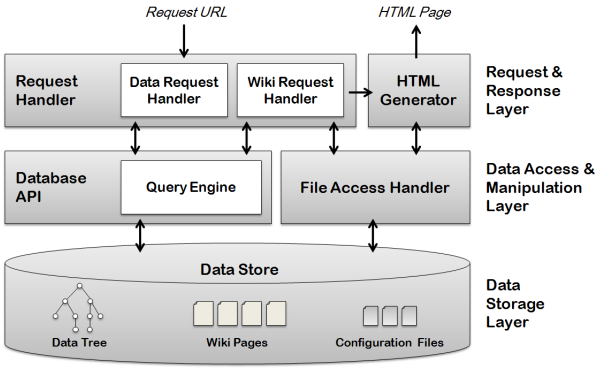


Figure 1: DBWIKI **System Architecture**

2. SYSTEM OVERVIEW

We implemented DBWIKI as a stand-alone Java application [12]. The architecture, divided in three layers, is shown in Figure 1. The bottom layer is responsible for storing the data and any additional information. The middle layer is responsible for querying and manipulating the data. The top layer of the architecture handles incoming HTTP requests and generates HTML pages in response. In parallel to the Java-based prototype presented here, we used a high-level Web programming language called Links [17] to develop another prototype, where we experimented with wiki-embedded query language design. Our experiences with the Links prototype are presented in a companion paper [16]; the lessons have been incorporated into the Java system.

2.1 Data Model and Storage

DBWIKI extends the XML archive management system XARCH [21]. XARCH is an archiving system based on a simplified XML data and schema model, as illustrated in Figure 2. The schema describes the set of possible paths in the data tree and distinguishes internal ($) from text (@) nodes. In XARCH, the edges of the data tree are annotated with time intervals indicating the range of times (or version numbers) during which a given subtree was present in the database. Timestamps are similar to those proposed in [14] and implemented in XARCH. For example, a timestamp $[1-5, 10-12]$ indicates that the associated node was present in database versions $1-5$ and $10-12$. A special value "now" indicates an open interval. XARCH supports an efficient sorting-based *merge* operation that identifies the differences between the current version of the database and a new version. XARCH has been used to archive real examples such as the CIA World Factbook [2] and IUPHAR-DB [7].

DBWIKI currently supports a common set of data and schema modification operations including insert, delete, and update. It is also easy to copy-and-paste

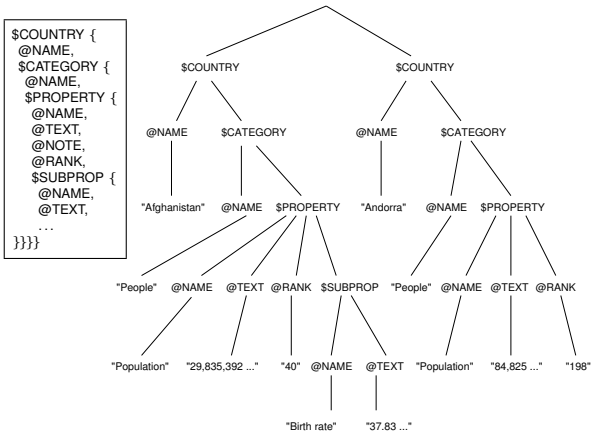


Figure 2: A schema and data tree

Figure 3: CIA World Factbook web interface in DBWIKI with time machine and edit menu

nodes and subtrees within or among different curated databases. Each operation creates a new version of the database (efficiently using the archiving approach from XARCH). With each node we associate a timestamp that describes those database versions in which the node was present, using the same interval annotation approach as in XARCH. Based on the timestamp and information about the action that created each database version we derive provenance information for data nodes following the provenance model defined in [11]. Each node may also be associated with a list of annotations. Annotations are timestamped but not versioned, and creating an annotation does not create a new version of the data.

We use a relational database back-end to store the data tree, annotations, and version information, *i.e.,* we shred the data tree, schema, and other metadata into relations. The data is stored relationally using the edge relation. Each tree is given its own entry number which identifies all the rows corresponding to that tree. Moreover, each row is associated with a timestamp indicating the versions when that edge was present in the tree. The main difference between the storage model of DBWIKI compared to XARCH is that instead of using XML files with special string-valued attributes to store timestamps, we store both the tree data and the temporal information in a relational database. This is useful for supporting annotation and provenance-tracking since we do not have to serialize this extra data into a textual XML form. DBWIKI supports different RDBMS back-ends using the Java JDBC interface. The relational database also stores the wiki page markup sources and configuration files used for web page layout (see below). Each of these files is also versioned.

2.2 User interface

Users interact with DBWIKI through a web browser, making requests encoded using URLs for either browsing or modifying the data, or for viewing or editing wiki pages. The URLs for wiki pages are similar to those in Wikipedia, *i.e.,* the page title is used as the page identifier. When browsing the data tree, we allow URLs similar to the query formats (see Section 3.1 for details). Once requested data has been retrieved, it is passed to the HTML generator that generates the response page. One of the design criteria of DBWIKI was to keep HTML generation separate from the rest of the system, hence highly customizable, as in typical wikis or content management systems. HTML generation is guided by three configuration files.

The first file is a HTML template. The template specifies the basic HTML to be used for all web pages of a data collection. Besides standard HTML tags, the template contains placeholders for predefined user-interface components. These components take the data node to be displayed and database version information as parameters and generate standard HTML snippets. One such component is the "time-machine". The time-machine provides links to different views of the data, *i.e.,* the current version, each previous version, highlights of changes since a given previous version, and the full history of the data. Other examples for predefined user-interface components are HTML and JavaScript code for displaying and editing annotations and provenance information. The second configuration file is a layout definition that specifies how to map the tree-structured data to HTML pages, tables or lists. The simplicity of the schema language makes it easy to specify different layouts for the data. The layout system allows configuring which attribute names are used in paths, whether groups are rendered as tables or lists, and how subtrees are grouped, *i.e.,* all on one page or split into several pages. The third configuration file is a cascading style-sheet (CSS) file used to format the HTML output produced by the tem-

Figure 4: Wiki pages (from left to right) embedded query result, page source, and data update form.

plate and layout engine.

The wiki interface also allows editing all configuration files on-the-fly. Together, these configuration files give the user a great amount of flexibility in customizing the look-and-feel of the web pages. An example is shown in Figure 3 which shows a DBWIKI version of the CIA World Factbook using some of the features of the original Web site. Figure 4 shows another three pages from the Factbook wiki using the default layout. However, while DBWIKI supports automatic browsing and editing of Factbook data, it cannot at present exactly duplicate all aspects of the existing interface. There is a trade-off between simplicity and expressiveness of the template and layout language, and we are interested in exploring richer stylesheet languages.

2.3 Database Queries

With DBWIKI one can query the data tree and embed the results in wiki pages. Thus, DBWIKI's wiki pages are dynamic, combining hypertext with views of the structured data. Wiki page queries are translated to SQL queries against the relational data store. We currently support three different query formats. The first format uses the node identifier to retrieve a node (and its subtree) from the database.

The second query format is a special form of path expression, *i.e.,* sequences of node labels with optional constraints. Path expressions allow positional references as well as constraints on values of a node's children. For example, the query `/COUNTRY:2` returns the second country in the CIA World Factbook [2]. Note that the order of nodes is defined by the order of their node identifiers which in turn reflects the order in which the nodes have been inserted into the database. The following query returns the population for Chile in our version of the Factbook.

```
/COUNTRY[NAME='Chile']/
  CATEGORY[NAME='People']/
    PROPERTY[NAME='Population']
```

The third query format (currently in development) is an adaptation of the XAQL query language that was implemented with XARCH. This format allows to select multiple sub-trees from a node as well as posing constraints on the timestamps and provenance information of nodes. For example, the following query returns the name and GDP of all countries that were modified by user admin since 2010:

```
SELECT $c/NAME, $p/TEXT FROM $c IN /COUNTRY,
$p IN $c/CATEGORY[NAME='Economy']/PROPERTY[NAME='GDP']
WHERE $c WAS MODIFIED SINCE 2010-01-01 BY admin
```

The path expressions in our queries correspond to a simple fragment of XPath, and we can use a wide variety of known techniques to evaluate them efficiently. We currently use a simple two-step approach: Based on the constraints in the path expression we first generate a SQL query that retrieves all candidate entries that potentially satisfy the constraints. We then load each candidate entry into memory and evaluate the path expression on it. For XAQL queries there is a third step to filter the results using the path expressions in the SELECT-clause.

3. TECHNICAL HIGHLIGHTS

In this section we give further details of the various components of the system. In this short paper we cannot give full details of algorithms but we refer when possible to similar techniques in the literature.

3.1 URLs

DBWIKI provides unique URLs for each node in the data tree. These URLs are essential for displaying and editing individual nodes in separate web pages, and for

node annotation. The URLs and the provenance information that we store, furthermore, provide a mechanism that can be used to cite (parts of) database entries and give credit to users for their contributions.

URLs in DBWIKI are based on node identifiers and they take the form *http://server/database/node-id*. While this URL scheme fulfills the need for persistent and unique identifiers, it is not very intuitive for human users, nor is it robust if data is exported and re-imported. In addition, DBWIKI is able to decode URLs that reflect the current path under which a node is located based on predefined node identification rules. For each group node in the database schema one can define a descendant attribute node whose value is to be used as node identifier. These identifiers are similar to keys for XML as defined in [15]. An example specification for the CIA World Factbook is:

```
IDENTIFY /COUNTRY BY NAME,
IDENTIFY /COUNTRY/CATEGORY BY NAME,
IDENTIFY /COUNTRY/CATEGORY/PROPERTY BY NAME
```

Based on this specification the following URL refers to the the population of Chile in our Factbook collection: *http://server/CIAWFB/Chile/People/Population*. To avoid ambiguities each path component can be prefixed with the element label, *e.g., /COUNTRY:Chile*. These URLs are not stable as the data and schema are likely to change over time; we plan to extend them with time information to make them more useful as stable citations [10].

3.2 Update operations

Updating the data. DBWIKI supports atomic insertion and deletion of subtrees and editing of attribute nodes. These operations were not previously supported efficiently in XARCH, which focuses on merging whole database versions instead. Each update operation is currently implemented by first loading the entry to which the update applies (including all past version information) into memory. We then perform consistency checks and translate the update operation into a set of INSERT and UPDATE statements to the relational database. Details for each type of operation are given below. We further record the system time of the update and the operation itself in the version table. We refer to the new database version by t in the following. In many cases it would be more efficient to translate tree updates to SQL queries instead of loading and storing the whole entry. We regard this as an opportunity for future work.

An insertion is handled in several steps. First, we ensure that the parent node is alive and check whether the subtree being inserted matches the schema. If not, depending on the database's policy we either reject the update, ignore non-matching data, or extend the schema. Next, we insert the subtree as a child of the parent. Each node in the subtree results in an additional tuple in the relational database. Only for the root of the inserted subtree we also explicitly store a timestamp $[t - now]$. Similar to XARCH, all nodes that do not have an explicit timestamp inherit the timestamp of their parent.

A deletion of a node in a given entry is handled by adjusting the timestamps of the deleted node and any live descendants that have explicit timestamps. Depending on whether the deleted node has an explicit timestamp we either replace "now" with $t - 1$ in that timestamp or insert a new timestamp for the node that ends at $t - 1$. We then replace "now" with $t - 1$ for all explicit timestamps of live descendants. Note that we do not delete any tuples from the relational database to ensure that all past versions of the data are available.

Each attribute can have multiple text nodes as children. We refer to these nodes as attribute values. A live attribute can only have one live value. When editing an attribute a we first terminate the current value of a. Let v denote the new attribute value. If a has had value v sometime in the past, then we extend the timestamp of the corresponding text node with $[t - now]$. If not, then we create a new child of a with timestamp $[t - now]$. We avoid performing (and recording) spurious updates that do not actually change the value.

Copy and paste. DBWIKI supports a simple form of copy and paste for data subtrees, following the design given in [11]. Copy and paste is implemented as an insert with the data to be inserted being retrieved using a DBWIKI URL. That is, one first selects the target node for the paste operation and then specifies the URL that points to the root of the subtree that is to be copied. This URL can either point to the current or a previous version of the subtree; in either case the value of the subtree at a single time instant is copied and pasted as a child of the target node. DBWIKI sends a copy of the subtree that is to be copied in XML format in response to a parameter "?cpxml" on the URL. The source URL of a copy and paste operation is recorded in the version table. Using URLs enables copy and paste between different servers.

Schema updates. DBWIKI currently supports only insertion and deletion of schema elements. The schema is versioned, so that past versions of the data can be understood. Insertion happens automatically when inserting data that require schema extensions. Deletion is implemented by deleting from the schema in the same way as for data deletion, and then deleting the corresponding data subtrees. We are currently investigating more efficient implementations of insertion and deletion as well as considering additional operations such as renaming.

3.3 Provenance and annotation

The initial provenance record for any database is the

record of when it was imported, by whom, and (optionally) a URL pointing to the source. Of course, we have no way of ensuring that the source URL is stable; it is just for documentation purposes. We also store one provenance record per insert, delete, copy-paste or update operation. Some of this information can be inferred from the change history. We also support annotations (as textual comments) to any part of the database.

4. EXTENSIONS

The choice of a very simple data model already pays some dividends, and suggests that we can select among a wide variety of techniques for querying and manipulating XML or tree-structured data, as well as for publishing relational data as XML. It should also be straightforward to export DBWIKI data as RDF/Linked Data, an approach to data exchange that is becoming popular in scientific data communities. Other techniques for XML such as security views [18] should also be easy to incorporate into DBWIKI.

A more sophisticated approach to path querying based on Grust et al.'s XPath Accelerator [19] has been implemented in a student project. However, the interaction between XML indexing and temporal issues remains largely unexplored. Another student has added facilities to query data and plot query results using charts, graphs, or maps, much as Google Fusion Tables allows plotting table data using Google APIs. For DBWIKI, this problem is more difficult because the data is nested. A third student has developed ways to visualize and query the provenance information, such as showing the number of edits or annotations per user over a given period as a chart or graph.

5. CONCLUSION

This paper presents DBWIKI, a system that combines the insights of wiki-style user interfaces with the capabilities to structure and query data efficiently characteristic of database management systems. The system is still under development, but we believe it represents solid progress towards the goal of making curated database technology available to those who need it the most: namely, scientific database curators and consumers of scientific data, where provenance, annotation, citation, and versioning are key requirements that currently need to be revisited for each new database. We also hope that DBWIKI will help us evaluate the effectiveness of research so far on these topics, and identify new research directions or unmet needs. The DBWIKI source code has been made publicly available under open-source terms [9], and we invite other researchers or projects to experiment with it, extend it or critique it.

Acknowledgments. This work has been supported by an EPSRC platform grant, by Google and by the University of Edinburgh IDEA Lab. Hui Li, Haoli Qu and Snehal Waychal have contributed code as part of their MSc projects. We are grateful to Tony Harmar and his colleagues for giving us access to the IUPHAR [7] source data.

6. REFERENCES

[1] http://semantic-mediawiki.org.
[2] https://www.cia.gov/library/publications/the-world-factbook/index.html.
[3] http://www.citizensciencealliance.org/.
[4] http://www.informatik.uni-trier.de/∼ley/db.
[5] http://www.geneontology.org.
[6] http://en.wikipedia.org/wiki/Help:Infobox
[7] http://www.iuphar-db.org.
[8] http://www.uniprot.org.
[9] http://code.google.com/p/database-wiki/
[10] P. Buneman. How to cite curated databases and how to make them citable. In *SSDBM*, pages 195–203. IEEE, 2006.
[11] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539–550. ACM, 2006.
[12] P. Buneman, J. Cheney, S. Lindley and H. Müller DBWiki: a structured wiki for curated data and collaborative data management In *SIGMOD*, demo, pages 1335–1338. ACM, 2011.
[13] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pages 1–12. ACM, 2008.
[14] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving scientific data. *ACM Trans. Database Syst.*, 29(1):2–42, 2004.
[15] P. Buneman, S. Davidson, W. Fan, C. Hara, and W.-C. Tan. Keys for XML. In *WWW*, pages 201–210, 2001.
[16] J. Cheney, S. Lindley, and H. Müller. Using Links to prototype a Database Wiki. In *DBPL*, 2011.
[17] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: web programming without tiers. In *FMCO*, pages 266–296. Springer-Verlag, 2007.
[18] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, pages 587–598, 2004.
[19] T. Grust, M. Van Keulen, and J. Teubner. Accelerating XPath evaluation in any RDBMS. *ACM Trans. Database Syst.* 29(1):91–131, 2004.
[20] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. in *SIGMOD*, pages 13-24. ACM, 2007.
[21] H. Müller, P. Buneman, and I. Koltsidas. XArch: archiving scientific and reference data. In *SIGMOD*, pages 1295–1298. ACM, 2008.

Meral Özsoyoğlu Speaks Out

on genealogical data management, searching ontologies, and more

by Marianne Winslett and Vanessa Braganholo

Meral Özsoyoğlu
http://zmo.cwru.edu/

Welcome ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today, we're in Providence, site of the 2009 SIGMOD/PODS conference. I have here with me Meral Özsoyoğlu, who is the Andrew Jennings Professor of Computing in the Department of Electrical Engineering and Computer Science at Case Western Reserve University, where she is also the department head. Meral's research interests lie in databases, bioinformatics, and pedigree data management. She is currently the Editor-in-Chief of ACM Transactions on Database Systems and a trustee of the VLDB Endowment. Her PhD is from the University of Alberta, in Canada. So Meral, welcome!

Thank you!

So first, I have to ask: what is pedigree data management?

Actually, perhaps a better term for it is genealogical data management, rather than pedigree. Pedigree is apparently more commonly used for dogs. But what we mean is family tree information, like parents, grandparents, ancestors, descendants. Pedigree data management is becoming very important in medical informatics. There are many publications and a lot of interest in using family history in clinical decision making. So pedigree, or, genealogical, data, especially for hereditary diseases, is very important, and all clinical institutions are collecting this data. There are not extremely large volumes of pedigree data currently, but it is growing very

fast. It is in the form of an acyclic directed graph, and, when you have large volumes of these acyclic directed graphs, querying them becomes an issue.

I am very much interested in query processing, making it scalable and efficient. So, I thought pedigree data management and querying would be a great topic to work on. Also, there are a lot of opportunities in medical informatics, and Case has a medical school where there are several life science researchers. So, think about a very large directed acyclic graph of genealogical information. Currently, this data is stored in the form of a relational database, and each patient, each person has mother and father information, and, in order to find ancestors having a particular disease, you have to do it by record at a time, which is not scalable for large volumes of data. So, we thought that if we have a special graph structure for pedigree data, and then if we link it to the actual record data, we can identify all the ancestor IDs, and then fetch that data right away as opposed to finding the father, and then finding the father of the father, or parents of the parents, and so on. In fact, we use certain kinds of encoding for this data, and then directly find all ancestors or all, let's say, second degree relatives, and queries like that, or find the inbreeding coefficient…

What's that?

Inbreeding coefficient indicates whether there is inbreeding in the genealogy or the pedigree of an individual. This is computed by finding the common ancestors. There are actually formulas for that. In fact, the formula for calculating the inbreeding coefficient that genealogists use is recursive. But computing it recursively, if it is large, is very time consuming. So we used a path counting formula for the inbreeding coefficient by just counting the paths to common ancestors; but then you have to eliminate duplicate (or overlapping) paths. And, this way, we have a significant improvement in the performance of calculating the inbreeding coefficient.

Another thing is kinship. There are these kinship coefficients. So again, we adopted the same path counting formula for the kinship coefficient, and we published that in the CSB conference last year, and then it was invited to the Journal of Bioinformatics and Computational Biology. So again, since there is not much real pedigree data that is large…

Even from animals?

Animals, yes. But we were collaborating with the Cleveland Clinic Foundation, and they have a hereditary colon cancer, called FAP. Apparently, Cleveland Clinic has the largest pedigree data collection on this disease. It is called Jagelman's registries, and it is the largest in the world. But then, when you look at it from a database researcher's perspective, it is not large at all. So we did our experiments on real data to show how scalable and how better performing our methods are, as compared to the traditional approaches. We weren't satisfied with it since the data was not very large. So we also generated some synthetic pedigree data using a PhD thesis from Iceland on this; where you can adjust parameters like the number of children, life expectancy, and things

like that, so you can get as deep a pedigree as you choose. And, even for very large pedigree data, our methods show significant improvement.

So for example, for that colon cancer database, how many different people do they have in there?How big is big in that domain?

I don't remember the exact numbers now. But I think it was something like hundreds of patients. Because what happens is that one person with the disease comes to the hospital, and, then they start to build the pedigree based on that individual, called progenitor. So they try to get the progenitor's parents, and, then, children and relatives. This is how they grow these family trees. If you think about it, it is hard to grow it with more than a hundred individuals for a particular family. So there are these isolated family trees which are not trees really: most of the time, they have lots of (undirected) cycles. Then, of course, these family trees occasionally merge as well.

We are also using similar techniques in bioinformatics for searching ontologies, for example, Gene Ontology, which we use in PathCase, a metabolic pathways database. Do you know what a metabolic pathway is?

I do, but maybe our readers may not all know.

There are a large number of metabolism-related reactions taking place for organisms, or humans, to live. Reactions have input metabolites, called substrates, output metabolites, called products, catalyzing enzymes, activators, inhibitors and so on. A metabolic network of reactions is viewed not as a graph, but a hypergraph. And the reactions of various metabolisms such as the carbohydrate metabolism or the lipid metabolism form a metabolic network. And, the network is really placed within a compartment hierarchy such as mitochondria inside cytosol, and so on. Genes relate to the metabolic network as gene products function as enzymes, and organism hierarchy also relates to metabolic networks as well. PathCase metabolic pathways database captures metabolic networks of organisms, humans included as graph database, and relationships to genes and organism hierarch as well. Users can zoom in and zoom out, viewing the metabolic network graph at different levels of abstraction. And they can query it, because it is a graph after all! You can search for paths, neighborhoods of a particular metabolite, or a reaction. You can also collapse complete pathways, such as glycolysis, into individual nodes and look at the network of pathways where the nodes themselves are pathways. This was the start of PathCase project, an NSF-supported project.

We are now working on the next phase of this project, called PathCase for Systems Biology. Original Pathcase assumed a static network. But then there is also the dynamics of it. Systems biologists model the dynamics of individual pathways, subnetworks of pathways, or just reactions using various kinetic models. These models are difficult to produce and verify. Nevertheless, their numbers are in hundreds to thousands in different data sources. Our new project maps systems biology models to/from the full metabolic network, and allows systems biologists to search, query, and visualize model and network information together.

So I see, so the main CS research issues that you took out of this biological problem had to do with the search and indexing aspect?

Exactly, and also the querying component.

Your encoding that you mentioned, is that like a Dewey encoding, or what kind of encoding is that?

(Yes. It is a prefix based encoding like Dewey encoding.) We call it nodecodes. In fact, a long time ago, when working in multimedia databases, we were dealing with searching acyclic directed graphs; so we used some sort of encoding of paths in such a way that, given two nodecodes for two nodes, you can find the paths between two nodes directly by looking at the nodecodes, not by traversing the graph. We used this idea for pedigree searching, with some differences. In terms of pedigrees each node has a biological mother and father, so you know that the in-degree of each node is definitely two. There are many optimization issues as well. For example, for pedigree, sometimes we collapse each family into a single node, that has a significant improvement on the performance.

How can a young computer science professor advance the state of the art in biology, while still building a strong reputation in CS?

This is a very good question! Currently I see many computer science researchers learning biology; at least many of my PhD students also take biochemistry; they also take bioinformatics, systems biology and computational biology courses. This requires significant time and effort. And, you need a really good collaborator. But again, there are a lot of funding opportunities and research challenges in the intersection. So I think collaborative research is very time consuming, but, at the same time, it is very rewarding.

What happened to all that work that people did on nested relations?

My answer might be biased, but I think all the work that was done on nested relations actually formed a foundation for XML, XML querying, and XML storage and so. In fact, today I just saw a demo in SIGMOD demonstrations that was utilizing relation nesting. So I think that is the nature of research: you work on certain topics and it becomes so popular that people keep producing results without truly knowing what will come of these results. And then, after some time, those results become useful and timely in another context, and people start using them. If you think about it, we now see datalog or rule-based systems research coming back; they were very active about 10-15 years ago; there was this quiet period; and now it is back.

You did your undergraduate and master's degrees in Turkey. Is it true that as many women as men get degrees in computer science, in Turkey?

I'm not sure if it is as many, but my sense is that definitely a larger percentage of women are going to engineering and sciences in Turkey than in the US. I did my sabbatical in 1991-1992 in

Turkey at Bilkent University, teaching the same courses I teach at Case. About 35-40% of students in Turkey were women, and, at Case, that was a much lower percentage.

So why do you think that is?

I think it is most probably because in Turkey there is a university entrance exam; everybody has to take this university entrance exam. And students are placed to colleges and disciplines based on their scores. Highest score areas are engineering and medical school (computer science is also very high), then comes sciences. So once a girl, a female, gets a high score, nobody questions whether that area is right for her or not. In here, one of my friends told me that "you know, I really liked math very much, but I didn't think going to computer science or sciences would be the right thing for a female". I never felt that way when I was growing up in Turkey. Here they always have the question: is this right for me? I never asked that question to myself.

Well, what about after you came to the US, did the different attitudes make you ask that question?

It didn't make me ask that question; it surprised me. I was the first women faculty member in my department, and the second women faculty member in the history of Case Engineering School at the time--we are talking about 29 years ago. Several people thought I was a secretary, and were asking me where the professor was, and I kept explaining them "I am the professor". But the attitudes surprised me. I didn't think whether this is right for me or not.

Have you been any other firsts?

I was the first female faculty member in our department. Then, when I became a department chair two years ago, I was the first female department chair in our department, and also the first female department chair in the history of the engineering school. I am also the first female editor-in-chief of ACM TODS.

Why are there so many Turkish database researchers?

I didn't think that there are too many.

Not too many, but a lot!

Still, more than what it used to be. I remember a PODS conference in 1983, I think in Atlanta. I remember some of my Greek colleagues teased me that there are so many Greek database researchers, but not many Turkish database researchers. So I am happy that there are now more Turkish database researchers! As far as I remember, the first Turkish database researcher who published internationally was Esen Ozkarahan. He did his PhD in Toronto on database machines. This was late '70s, I think. He is the first that I know. And then, I think, me, Tekin (my husband), and Tamer Ozsu. And there are now many younger Turkish researchers: Ugur Cetintemel, Selcuk Candan, Fatma Ozcan, Nesime Tatbul, and many others. That's very good.

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

This is a good question. I will say this: databases is a great field because data is everywhere. So it is a never-ending… You can always find challenging problems. I remember a time where, in database conferences, we were thinking "oh, this is the end of databases; all the problems are solved". But it is not. So I would say, just keep on looking for challenging problems, and the ones that you like--that is very important, you want to enjoy what you do. There are many challenging problems to be solved.

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

From my research?

Yes. Something you produced, and if you don't have a favorite, that's fine too.

I think my favorite piece of work was determining tree query membership and query optimization using semi-join reductions. That was my very first paper. I was a PhD student, and I published it in COMPSAC conference in 79'.

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

As a researcher?

Well, I could say, as a computer scientist.

I think I would like to be better at coding. I wish I was a better programmer.

Thank you very much for talking with me today.

Thank you!

Divesh Srivastava Speaks Out

on the importance of looking at real data, abstracting problems and more

by Marianne Winslett and Vanessa Braganholo

Divesh Srivastava

http://www.research.att.com/~divesh/

Welcome to this installment of ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Shanghai, site of the 2009 International Conference on Data Engineering. I have here with me Divesh Srivastava, who is the Executive Director for Database Research at AT&T Labs Research. Divesh's research interests include logic programming, OLAP, data streams, XML, data quality, and data anonymization. Divesh claims that he personally has no honors, but is the boss of all those AT&T researchers who've won many prizes, including best paper prizes at ICDE 2009 and VLDB 2008. His PhD is from the University of Wisconsin Madison. So, Divesh, welcome!

*Divesh, AT&T is a phone company – in fact, it used to be **the** phone company in the US. What do databases have to do with the phone company?*

I can say two things. First, AT&T used to be a phone company, but it is now much more than a phone company. Think of it now as a communications company, or a networking company. In the phone network, there is just one kind of network, and the operation of these complex

networks require a lot of data, right? In the operations, you can generate a lot of data, and to understand how the network is operating, you need to do a lot of data analysis. You need data analysis also, even for understanding how to design and how to evolve the network, whether it is the phone network, the IP network, the wireless network, right? And all of these things require a lot of data. So because of that, over the years, AT&T has had a history of collecting and analyzing various kinds of, I guess I should say packet-header data, for there are many kinds of networks, and because of that, there is a history of both collecting and analyzing large amounts of data. Trying to model what's in the network requires extremely complex databases. So in that sense, databases are almost critical to AT&T.

Then, maybe I should ask you what you can tell our academic audience about real databases that they don't already know.

I guess real databases tend to be very complex, as well as very large. *(How complex? How large?)* In AT&T, we don't necessarily have a single database for everything. They just evolve over time, so there is a lot of legacy, but we have even single databases that may easily have upwards of a 1000 tables, 30-40,000 columns in the database, and it is not your employee department database, it is not even a database where you can put the schema on a piece of paper and stretch it out over the floor of this room that we are sitting in, and even understand it at that level. So it is very complex. At the same time, we also have other databases that are extremely voluminous. So we have databases where we have close to a trillion records. That is huge. We have databases where we are collecting data in a streaming fashion, and it is so huge, that we can't even afford to store all of it. So that is some of what happens in AT&T as an example of a real database.

What about data anonymization – how does that tie back to AT&T's interests?

It is a recent interest. I cannot say that I have a complete understanding of how all it may be used for AT&T. But one of the interesting things, and maybe this is of interest to you, Marianne, given your interests as well, is often for reasons of regulation or law, we are all still… say, okay, we have been collecting this data for a particular purpose, for the network operations, but maybe it contains a lot of sensitive information. So you can only retain it for a certain amount of time. This certain amount of time might be 6 months, a year, 2 years, depending on the nature of the data. But given that we are in research, we aren't interested in necessarily violating anybody's privacy, right? But we do want to understand trends, and trends are often over much longer time periods than the 6 months that is needed for the operational use for that data. So I would like to be able to say, can I take, let's say, 10 years' worth of data, and anonymize it in a way such that it meets the legal requirements for the auditors, that we are not trying to preserve data that might violate somebody's privacy? But there are enough trends and statistics buried in the data that I can use it for analysis. So I would like to be able to anonymize data over a very long time period, keep it so that I can use for interesting analysis, that I may not even have thought about today. So that is one of the reasons why I think AT&T should be interested in data anonymization. But as I

said, it is a recent research interest of mine, and how the techniques that we are using today actually work to solve this problem? I still don't have a good answer yet. Is that fair?

That's fair! That's fair! You were working on streams before almost anyone else. Looking back at what we've accomplished in that area, where do you see that work having the most impact?

Certainly that is true among the database community. But there were people working on streams from the theory and algorithms community way before. Munro and Patterson's is from (I think) 1980[1]. And that is one of the first papers that talked about doing certain computations in a streaming fashion. And at that point, nobody had thought about streaming as a viable technique. But I think this goes back to one of the questions that you asked me earlier: AT&T has huge amounts of data, and can collect huge amounts of data, and because they realized that we cannot necessarily store all this data while still people are interested in analyzing this data all the time, that's why we got into streams in the first place. And so, where we got into streams, I think it was in 2000, 2001, around that time, is where we are in data anonymization today, right? We see a need, we begin to sort of understand how it might be useful, but over the last, close to a decade, we have done very interesting research in terms of core foundations for streams, we have also built data streaming management systems, probably among the most efficient there are. Because we have to be able to analyze data at the rate that data is moving through the network.

So has that technology made it into production systems at AT&T?

Yes, so AT&T's data stream management systems are tools that we have developed and are something that are being used on an operational basis in various parts of AT&T's network. It is used in some parts of the core network, and because of that we have to deal with networking speeds, that are like OC192, OC768, which are sort of at the leading edge, right? Try processing, something like a million, ten million, 50 million records a second…

Not with my hardware…

So in this case we are using, not necessarily massive souped-up hardware, but this is sort of well designed hardware, but also very well designed software. We are talking about this from a database point of view. Interestingly, this research that we are talking about (data stream management systems), really came about because there was a need in networking. And the database people had the technology, the ideas. So it is really a collaborative effort, between the database and the networking folks. And actually, that is a common theme at AT&T, right, we like to collaborate. We can't say that database people, sitting in their rooms by themselves can solve all the data management problems. Collaborating with the networking folks was critical. They couldn't have done it by themselves, we certainly couldn't have done it by ourselves, but happily, getting the two groups together, we did something.

[1] J. I. Munro and M. S. Paterson. **Selection and Sorting with Limited Storage**. Theoretical Computer Science, v.12, pp. 315-323, 1980.

So now you are working on data quality. Is that going to be the next hot topic?

I hope so. I think data quality is one facet of something else I touched upon, which is, we are dealing with extremely complex databases, and some of these complex databases are not complex on day one. They are complex because they have evolved over a very long time period. They are complex because they are trying to capture aspects of reality, in this case networks, which are extremely complex physical entities, and over time, the people who used to know about some parts of the database because they had applications that use that data have gone away. Expertise disappears. So nobody has that (sadly) complete understanding of complex databases. So I think one of the big challenges is really dealing with an understanding complex evolving databases. And data quality to me is one facet of that, because, over time, people have data, and it gets added, and when people don't necessarily understand it, it is very hard to know when is the data correct, when is it capturing all the aspects of reality that you were hoping it captured. And so we have been looking at sort of different ways of dealing with data quality.

At this point, with data quality, it is like the blind men and the elephant story. You know the story of the elephant? There are a bunch of blind people looking at it from different facets, and thinking they have understood it completely… That is where we are, right? I think we have at least a half a different ways of thinking about data quality. So one of the simplest ways is sort of a sense that says, if I can understand the semantics of the data, if I can abstract it, not by sort of gathering requirements or something from humans, but just looking at the database itself, and statistically analyzing it to see what can we say about semantics, from a statistics perspective, from a distribution perspective, and things like that. Maybe then we can abstract that into some kind of semantics, integrity constraints of some kind. And then we can say, well, things that don't match the integrity constraints, maybe they are potential quality violations. But the way humans work, it is much easier for us to say, here is what I came up with as a semantics, here is what I think is a violation, you as a human who is an expert on this domain, can you tell me if this makes sense? And it is easier for them to answer yes/no, rather than just asking them, can you sit down and summarize what is the semantics of this database? It is not feasible. So that is one way of looking at data quality.

Another recent way we are trying to understand data quality is, people used to think of quality as something where dirt gets into the database, right? And people have all kinds of ways of trying to prevent anomalies. But we are trying to sort of take that one step beyond and say let's try to understand what kind of updates we could make to databases. Not all of the kinds of errors that creep into databases can be captured via integrity constraints. Even SQL statements, as complex as you want them. Are there better ways of trying to prevent dirty data from getting into database? Maybe I intended to make a certain update, I made a mistake in my thinking, I specified something different, it doesn't seem to violate any constraints, it gets into the database, and now it's in there. Are there ways to sort of deal with that? So those are sort of positions of two blind men, I don't think we are anywhere close to understanding all the ways in which one can think of quality in data semantics. But in some sense, that is one of the grand challenges. We

need to make sure that the data that people have in databases is something that is of good quality, something they can trust, something they can rely on, because otherwise, people will stop using the data, and then it will deteriorate even further. So that is why I think it is important.

I have been told that you "think like a theory person", meaning that you are good at abstracting problems.

I hope so!

Don't people tend to define a problem to be what they can solve?

Yes, but hopefully, as researchers, we constantly learn, and constantly learn of solving things in new ways. It may be true that how I abstract things today is a function of what things I think I can solve today, or have some idea of how to solve today. But over the past 15 years, I've learned many things. I am sure that how I abstract things today and how I abstracted things 10 years back are not the same.

You think not?

I hope not, because otherwise I have wasted 10 years of my life!

Maybe, but you are abstracting from different problem areas. Now you are abstracting in data anonymization, and before you were abstracting in…

Sure! But, you know, being in a research community, constantly interacting with people… Again, I don't do this by myself. I am not in research because I like to sit in an ivory tower. I am in research because I like to collaborate with people. And I collaborate with new people, and they bring in new ideas, new techniques, right? So most recently, for instance, oh, not most recently, but as an example, I have been working with some people who have a lot of expertise in information theory. Ten years back, I did not have that expertise. Now I have a better understanding of it, so certainly, there are some problems that I abstract today that can make use of the information that I have, the knowledge that I have in information theory. Ten years back I didn't have that. There was no way I could have abstracted things in the same way.

That ties into something that several people mentioned to me, they said that although you worked with many different people and juggle many different projects, every time they meet with you, you give the impression that you have thought of nothing but their project since the last meeting. Several people also said that you pack 48 hours into every 24. So how do you do that?

Compression, I don't know! I enjoy doing research. To me, it is not something which is onerous. So, if you enjoy doing something, you think about it all the time. I don't turn off my research thinking. Was it Stefano Ceri who was saying he likes to think when he jogs? In the invited talk today? Well, I like to think while I am cooking. I like to think while I am walking. But I think, maybe the question came from people who actually do far more work on the projects than I do,

and I just provide maybe an insight or two, which they missed for some reason, and that may give them that impression.

So, since you have worked in many different areas, can you suggest a good methodology for choosing research topics?

So I can tell you how I choose a topic, and maybe that will work for others, and maybe it won't. But I have the good fortune to be in a research lab, right, where I have access to both very smart academically oriented researchers and colleagues, but I also have access to the industry, right? So I have access to people who generate the data, so I don't need to necessarily invent or make up applications. And having access to data, right, I think that is critical. There is no way I could have done a lot of the data streaming research or even the recent data quality research that I am doing without access to the data. This may sound a bit strange, given how we think of how databases are to be modeled, but I looked at some databases at some point in AT&T and we found database columns in a particular table that in the same column you may have a phone number, email addresses, what look like login ids, right, and you sort of say, that is strange, right? That is not the way people design databases. But it turns out that the column was being used as a user ID column, okay? For internal AT&T employees, and this was a database that came about over a long period of evolution merging data that was being used for different applications. And some applications had been using somebody's phone number as an identifier, which it is, right? Somebody else had been using somebody's email address as an identifier, and you merge these databases, you create an integrated database, and suddenly you have all these kinds of fields of information in the same column. And so we thought of, you know, this is a problem where you don't design databases this way, necessarily, right? But you come up with something, and now you suddenly have a lot of heterogeneity in the database. Can you even understand techniques for identifying heterogeneity? So we wrote a paper about it.

But without access to the kind of data that ended up there, having to imagine this, and if I were to simply say imagine that this happens, I am sure some reviewer would come back to me and say, you know, why would this happen? But know, instead of why would this happen, I know it is there. So having access to data in some sense is an unfair advantage I have.

Well, I think being in an industrial research lab, you should have some advantages that come from that, it seems fair to have some advantages.

I agree, I am not complaining!

You mentioned to me earlier that Bell Labs, of which AT&T Labs was once part, has many distinguished alumni.

Including yourself Marianne!

Thank you Divesh! So why are there so many alumni, but you've been there your whole career?

How do I answer that? I guess AT&T/Bell Labs have been around for a very long time. And certainly, even the database *alumni* have been around for a very long time. So people started, I would imagine even in the 1970s, right? So certain people like Rakesh Agrawal joined there in the early '80s, Jagadish was there in the mid-80s… and a lot of very good people… And they stayed for quite a bit of time. Jagadish was actually the head of the database group before I became the head. And he was there probably close to 14 years. I don't remember exactly how long Rakesh Agrawal was there before he moved, but it must have been at least I would say 7 years, or something around that time. I was even there when people like Carlo Zaniolo used to be at AT&T. But in some sense, I think of it as there is a lot of good people, right, and over time, maybe their interests change, maybe some people wanted to try out being a professor, some people wanted to have an impact on a database vendor, and so they moved over time. And I have been fortunate in having a lot of good colleagues that I continue to work with, so I am happy where I am.

We see a lot of students interested in database research coming out of IIT-Bombay, which is your own alma mater. How did that happen? How did that tradition get started?

I guess when I came out of IIT-Bombay (this was in 1987), and I went to grad school, at that point, I personally didn't have any sort of set view that I wanted to do databases, and I hadn't done any database research at that point. So I came to grad school in Madison, and Raghu joined as a faculty that same year, in '87, and then I took courses with him, at some point, and got interested in databases. But what happened at IIT-Bombay, much later was, there was a concerted attempt at recruiting people back from the US, right, and I think they made the good realization that rather than get one person, one person, in sort of a dozen different areas, which doesn't give people sort of a community, they decided to hire people in specific groups, and one of the first groups they chose, either actively or sort of fortuitously, happened to be databases. So they hired Sudarshan, who was my academic brother, right. He was also a student of Ragu's, who joined actually Bell Labs, after graduation, but then decided to go back to India. So even though Sudarshan was from Madras, which is a different city, he went back to IIT-Bombay.

Then they hired, I think, Sunita Sarawagi and Soumen Chakrabarti from Berkeley, right, both very strong, very good, database, some IR, some web, they went back. Then more recently, Krithi Ramamritham went back. And so it became like starting from a nucleus. It grew, and at some point I think they will probably have the strongest database research group in India. And so when you have such a good group of people, who are active in the research community as well, it gives an impetus to new students from there to continue that line of thinking. I think that is how it happened.

I heard that before you left Bombay you had never experienced temperatures below 75 degrees Fahrenheit. Did you realize what you were getting into when you left for Wisconsin?

Not in terms of the weather, definitely. This might be sort of partly incorrect memories, but I think the brochure that the University of Wisconsin sent, and you can tell me if the University of Illinois does this right, but all the pictures in the brochure, the glossy pictures, was of sun shine, people sitting on the lake, people walking on State Street. There was nothing much about sort of snow or 2 feet of snow and people sort of huddling in jackets. So of course I had no idea, right? But I sort of enjoyed the first winter I was there, it was fun. It was something new, and the cold still doesn't bother me. It wasn't expected, but it wasn't unpleasant.

I spent a month this fall at Infosys in Mysore, and you look more traditionally Indian than anybody at Infosys. How did that happen?

So tell me how do I look more traditionally Indian?

Well, you have more hair, you have more beard, and then you've got the different kind of shirt, and all the guys there were wearing pants and shoes like yours, but then they were wearing western style shirts and had western style hair. So they've all gone western.

Well, I don't think that is any different. I don't know if you have heard this, but when I came to the United States, I think my hair was probably this size (shows a very small size with his fingers). (*Ok, that's how they are now.)* So I don't think I was that different in India. I just happened to say, okay, I can grow my hair, let's try it out. And I liked what it looked like.

What about the Indian shirt?

I think it is very comfortable. It is very pretty, and periodically my parents, when they come, ask me what they can bring for me.

So they are supplying you with …

So they are supplying me with these things, and I like wearing them, so I wear them. Am I more traditional? I have no idea. I think there is a range of people. Or maybe I am just sort of still in 1987 when I came from India, and people in India have evolved.

Could be…

Could be.

So people describe you as "sphinx-like", with an "enigmatic smile". They say that it is hard to tell what you are really thinking, and that you don't reveal your strong opinions. How has this been helpful in your career?

Is this true, from your interview?

Well, I didn't know you were like that, but when they said it, I could see it. They know you better than I do. So when they said it, I could see that that was consistent with what I know. Because, for example, I didn't know what things you had strong opinions on.

I don't have too many strong opinions on too many things.

Well, I don't know, they said that you do, but you tend not to reveal them.

Maybe that is why I am a manager today.

Is it a useful characteristic for managers?

Sure, at least in some ways, right, I mean, there are different styles. People collaborate with people. I think I like my interactions with people to be pleasant, rather than adversarial.

In meetings, don't you ever have to convince people of something, say that streams are an important direction for AT&T or something like that?

No, because very often they realize that themselves. With my colleagues, for instance, people with whom I work, they don't work for me, they work with me. And if I try to tell them what to do, they would get up and walk out, which is exactly what they should do. If I am not able to convince them as opposed to just tell them what to do, I don't think that would work in a research environment. We are all very head strong. Nobody likes to be told what to do.

I can think of some people in the community who don't fit that mold though. But I am not going to name any names.

Can you tell us about the Rattlesnake Ranch?

So, where did you hear that from?

Oh, one of your colleagues, I actually don't remember which one. But if I did remember which one, I wouldn't tell you.

It is one of my favorite restaurants in New Jersey. So I like spicy food, very spicy food. The Rattlesnake Ranch buffet was introduced to me by a friend because it has a dish called the Chicken Habanero, where it is sort of extremely spicy, right, in a single chicken dish, maybe they have a couple of whole habanero peppers, sliced up and cooked with it. I like spices, so I like going there. They also have a kind of Margarita, which is a Blue Curaçao margarita, sort of a bluish color. So I like going there. And it is sort of an out of the way of beat place. They have a sign I think on the entrance, which I haven't seen anywhere else, which says, men are not permitted if they are wearing sleeveless shirts. It has a strange character to it. The food is very good. It is close to AT&T. I like it, and I often take a whole group or the whole department there for a department lunch or something.

Do they serve rattlesnake there?

Occasionally, they also have alligator, ostrich, bison, all kinds of stuff.

Ostrich is supposed to be the next big meat.

It is very tasty. I tried it a couple of times. I can't say I am a big meat person. But I like it, it is nice. Isn't kangaroo supposed to be the next big meat now?

I didn't hear that.

I think the Australian government is trying to promote that. Who knows if it is good or not…

As long as we don't start raising them here. I can imagine. My garden is already over run by rabbits and squirrels, imagine if kangaroos where popular also.

Your barriers around your house wouldn't work, right, they would just jump right over them.

Good point. The rabbits are deterred by very small fence, but a kangaroo certainly would not.

Do you have any words of advice for beginning or midcareer practitioners or researchers?

Well, I think, you know, whatever is fortunate enough to be involved in, right. Being in a place where you have access to real data, right, I think is very important. Twenty years back, which is when I started my research career, sort of, it was hard to get access to real data, a lot of real data. Today, at AT&T, I have access to AT&T specific data, but I think it is not that difficult any more to get access to lots of real data. People get access to data from the web, you can get access to lots data, lots of real data on the web, people can build crawlers. So I think one of the important things if you want to do database research, which is both research and practically relevant, is to look at real data, and look at lots of real data, because you can find things in real data that you won't be able to imagine.

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

I guess even that has changed over time. I am very fond of some of the recent data quality research I am doing. But that may simply be because I am doing it more recently. I certainly was very fond of the data stream research I was doing. I think even at this conference I have a data stream paper, which is a very cute paper. It's sort of is a paper on data streams dealing with decayed computations over time.

What does decayed mean?

It means for instance, if you want to be able to look at a stream of data, and some of the older data is still relevant, but less so than the recent data, so the importance of a data item sort of reduces over time as it gets older. For the longest time, people have realized this right, so they have come up with exponential decays and polynomial decays and things like that. It turns out

that in a data streaming system apart from exponential decay, people proved the hardness of all the other decays. It is very hard to do heavy hitter computation, which is needed for doing anomaly detections.

So you have this mismatch, well the mismatch said was, people realize the importance of having decayed computations, at the same time, the theoreticians, and maybe I am part of that community, sort of went ahead and said, doing it this way was very hard. So what we did was we sort of flipped it around and said, maybe you are thinking of decayed computations in one way, and there are alternative ways of thinking about it. So we came up with what we called forward decay as opposed to what people had been traditionally thinking about as backward decay. And then we showed suddenly with forward decay, it makes sense in a lot of contexts. Thinking about exponential decay in a forward or backward fashion is the same thing. Thinking about polynomial decays in a forward fashion is often more intuitive than thinking about it in a backward fashion. But you could suddenly do decayed computations with exactly the same efficiency you could do undecayed computations. Interestingly, even for exponential decay, you could do things like sampling, like reservoir sampling. And people had been trying to do it, and they came up with only partial solutions. But just flipping the way you think about it, thinking about forward decay as opposed to backward decay, suddenly there was a complete solution. It is a very cute problem, just a different way of thinking, which resulted in very interesting results.

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

Probably learn more what other people are doing. So, in AT&T Labs, we are fortunate to have colleagues in a wide variety of disciplines, networking, visualizations, statistics, software… take your pick, right, speech. I am sure that if I knew better what the other groups were doing, I could find more interesting ways of interacting, of combining their application needs with what the database people know how to do, or learn new things of how to do. If I had more time, I would certainly like to do more of that.

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

As a computer scientist, it is very hard say. Not too much. I luck into databases, thanks to Raghu. I think databases is one of those fields which has only become more important over time, everybody's data needs is only increasing. I don't know anybody who comes and says, you know, my database needs to be smaller than what they were a few years back. So I happen to be in an area of computer science where the research problems, the challenges, the needs, are only becoming more important over time. I am happy doing databases, and hopefully will for a while.

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

Thanks, Marianne.

Data Management Research at NEC Labs

Data Management Research Group
NEC Laboratories America
http://www.nec-labs.com/dm/
{hakan}@sv.nec-labs.com

1. INTRODUCTION

In 2009, NEC Laboratories of America started a data management research department to create a world-class team and a research program with the dual goal of research excellence and direct contributions to the company's global business. Our organization gives the researchers opportunities to maintain a balanced mix of fundamental and applied research as they focus on innovations which are motivated by the real needs faced by the company's large service and product businesses. In this report, we present an overview of the research program of the Data Management Research group, which currently includes Yun Chi, Hakan Hacıgümüş, Wang-Pin Hsiung, Bin Liu, Ziyang Liu, Hyun Jin Moon, Oliver Po, Jagan Sankaranarayanan, Junichi Tatemura, and our close collaborators, Michael Carey from University of California, Irvine, Hector Garcia-Molina from Stanford University, Jeffrey Naughton and Jignesh Patel from University of Wisconsin, Madison. The group is also engaged with numerous academic organizations through our University Relations program.

The current focus of the group is data management in the cloud. Cloud computing has emerged as a promising computing and business model. By providing on-demand scaling capabilities without any large upfront investment or long-term commitment, it is attracting a wide range of users. It is obvious that cloud computing presents challenges and opportunities for data management services. For instance, database service providers may face heterogeneous customer workloads with widely varying characteristics. To serve such workloads, they may have to use a diverse set of specialized database products and technologies in an elastic manner to ensure that customers observe the benefits of those products specifically tailored for their needs. The goal of our group is to understand and analyze those challenges and opportunities by identifying and solving the relevant research problems.

2. THE CLOUDDB PLATFORM

Our current research projects are built around a large platform, called *CloudDB*, which is envisioned as a comprehensive data management platform in the cloud [2, 10]. *CloudDB* would provide data management capabilities as a service to transparently and efficiently support diverse application workloads with identifiable SLA guarantees and end-to-end system management functions. The system would be able to employ heterogeneous underlying storage models to effectively meet applications query and scalability requirements. The CloudDB platform has the following guiding principals in its design:

- *One size does not fit all*, hence the platform should embrace the heterogeneity by leveraging specialized database technologies to serve diverse business needs and workloads.
- The *profit (money)* and the customer *Service Level Agreements (SLAs)* should be the main metrics for all system management and optimization decisions.
- The system should maintain the declarative nature of data processing while leveraging diverse set of specialized database technologies underneath.

The effort to build the *CloudDB* platform with these guiding principles has enabled us to identify numerous challenging, exciting, and relevant research and system problems in data management. Our progress in building the CloudDB platform has already generated various significant technologies, which are being evaluated, further developed, and deployed in real business settings. As the CloudDB platform includes a large number of projects, here we only report on the selected subset, which is a good representative of major areas including query processing and optimization over heterogeneous data stores, intelligent resource and workload management for

cloud database services, and application areas that leverage the cloud data management capabilities.

3. MICROSHARDING: ELASTICITY FOR OLTP WORKLOADS

The goal of our Microsharding project is to establish a declarative approach to support OLTP type of workloads elastically based on the relational model and to claim that developers do not have to abandon SQL and relational models just because existing RDBMSs are not as elastic as key-value stores [14]. Cloud computing is expected to enable an application to dynamically adapt to growing workloads by increasing the number of servers. Such an approach is often called *scaling in/out*, and the property of systems that enables this is called *elasticity*. Elasticity is an important property for hosting web applications since workloads from web users are often unpredictable, and can change dynamically over short periods of time. However, it remains challenging to deliver elasticity to interactive and data-intensive applications that handle a large number of read and write operations on shared data.

An approach to deliver elasticity to OLTP workloads is to use a family of key-value stores, which enable seamless scaling out (e.g., live data re-partitioning). However, these data stores provide limited query and data manipulation APIs that are much simpler than SQL. Whereas SQL provides a declarative way to query and manipulate data, those APIs require an application developer to code data manipulation logic in a more procedural manner. Such an approach lacks data independence: Change in data organization on a data store (e.g., introducing secondary index objects) involves change in the application code that access the data.

The Microsharding proposes a relational alternative to the industrial state-of-the-art approaches. We believe that the idea of entity groups [3] is practical and effective to achieve elastic OLTP workloads. If the workload and entity groups are designed appropriately, the system can mostly avoid distributed transactions, and scaling out could be made easier. However, the code that accesses data is written in a proprietary and procedural way, making it difficult to take a principled approach to design elastic workloads. Developing such a declarative and principled solution is the main focus and the novel contribution of microsharding.

To realize a relational approach as an alternative for the existing procedural approaches, we identify the following key questions: *1)* How can we define constraints on transactions similar to entity groups in the relational model? *2)* How can we benefit from the relational model to design and analyze elastic OLTP workloads? *3)* How can we implement it?

For the first question, we propose to extend the concept of transaction class [5], which is a description of the behavior of transactions. Our vision is to introduce transaction description language (TDL) to describe various restrictions on ACID properties in the form of transaction classes. We first introduce a primitive transaction class, which is a building block of complex transaction classes. A primitive transaction class defines the smallest unit of logical data partitioning, which is a relational version of entity groups. We call this logical partition a *microshard* and the entire scheme *microsharding*.

For the second question, we examined how TDL enables a principled approach to design databases and application workloads. The physical design (data layout) of the database takes not only schema into account but also transaction classes.

For the third question, we have implemented microsharding methodology as a relational middleware, called *Partiqle*, on top of open source distributed data stores, HBase [1] and Vodemort [4], which demonstrates the benefits of the data independence.

Transaction Class is a key concept in microsharding. We introduce a *transaction class* as a way of declaratively specifying the constraints on the ACID properties of a transaction. The concept of transaction classes was introduced to a distributed database system SDD-1 [5], where a transaction class is to specify data set accessed by a transaction. A transaction class was used as an input of static conflict analysis to optimize transaction protocols without sacrificing global consistency. Here, we use a similar specification to restrict the power of transactions the application can use. Our transaction class is a tool for the developer to design trade-off between consistency and elasticity. Here, we only discuss a basic transaction class, called *primitive transaction class*. More detailed discussion on transaction classes can be found in [14].

A primitive transaction class defines a logical scope of the data where serializability must be maintained. Its specification looks similar to specification of entity groups. Let us take an example from TPC-W benchmark data. Consider supporting a transaction that updates a purchase order, which consists of one record in `ORDERS` table and records in `ORDER_LINE` table, which have parent-child (foreign key) relationship. A transaction class for this transaction can be stated by specifying as follows:

```
CREATE TRANSACTION CLASS t1 AS
    ORDERS BY O_ID, ORDER_LINE BY OL_O_ID
```

This statement classifies records of `ORDERS` and

ORDER_LINE together into groups by the values of O_ID (the primary key) and OL_O_ID (the foreign key to ORDERS), respectively.

We call columns of a table specified in a primitive transaction class *transaction keys*. A value of this key identifies a specific group of data.

This specification is similar to data partitioning in RDBMSs based on reference [7]. The objective of traditional data partitioning is to provide physically partitioned but logically seamless (consistent) data. However, our partitioning defines *logical* boundaries that have an impact on the data consistency semantics. In addition, this logical partitioning is independent from physical partitioning (or layout on distributed data stores). To distinguish this difference, we refer to the logical partitioning as *sharding*. We call the unit of transaction scope a *microshard* and our approach *microsharding*, since the granularity of shards is very small relative to the entire database [1].

4. MAESTRO: RESOURCE AND WORKLOAD MANAGEMENT FOR CLOUDDB

The Maestro project aims at developing a family of technologies to manage very large heterogeneous database clusters that are deployed in cloud service delivery infrastructures. More specifically we look into resource and workload management and optimization based on the key metrics that are relevant for cloud service delivery. Obviously managing the resources and the workloads in a cloud service delivery infrastructure in an optimal manner involves a number of decisions that have to be made in the lifecycles of the systems. The key issue is the identification of the metric on which the system optimizes. The metric should be relevant to database systems characteristics and the cloud computing model. We choose to use *SLA-based profit* as the metric, which is the ultimate goal of service providers, rather than low-level system metrics, such as average query response time. SLA-based profit is identified by two parts: i) the *revenues* (money) and ii) the *operational costs*. The revenue is what service clients pay to the service provider based on the delivery of the services according to the SLAs in the contract between the service provider and the customer. The revenue is not fixed and it may change with potential reduction in payments or even penalties; depending on the service quality, e.g, too high query latency. Operational cost is the cost of resources used to run the service. Hence, the research question in Maestro is how to manage resources and workloads in the system to maximize SLA-based profit, which is SLA-based revenue minus operational cost. We believe applying SLA-based profit optimization to all system components opens up many interesting research challenges and opportunities.

In the Maestro project, we have been developing techniques to achieve our goal of *SLA-based profit optimization* in key areas such as query dispatching, query scheduling, admission control, capacity planning, provisioning, and multitenant database management. There have been many proposals to these problems in the literature. However, most of them addressed the problems by considering lower level system-oriented metrics, such as minimizing average response time or minimizing average slow down. That is, they do not explicitly consider profit optimization. It may seem like average response time minimization will implicitly lead to profit optimization, but it is not the case in general: some jobs may have more expensive SLAs to violate than others, so the cloud provider would want to give a higher priority to those jobs for better overall profit. Our approach, in contrast, explicitly considers SLA revenue function of each job at the core of those listed problems to achieve an overall profit optimization. We experienced that distinctively optimizing the individual system components that correspond to those key areas with a global objective in mind gave us a greater degree of freedom to customize our methods. This approach yielded higher degrees of performance, customizability based on variable business requirements, and end-to-end profit optimization.

The Profit and the SLA Models: The total profit, P, of the cloud service provider is defined as $P = \sum_i r_i - C$, where r_i is the revenue that can be generated by delivering the service for a particular job i and C is the operational cost of running the service delivery infrastructure. We define the revenue, R, for each job class in the system. Each client may have multiple job classes based on the contract. We use piecewise linear functions to characterize the SLA revenue as discussed in [6]. Intuitively, the clients agree to pay varying fee levels for corresponding service levels delivered for a particular class of requests, i.e., job classes in their contracts. For example, the client may be willing to pay a higher rate for lower response times. This characterization allows more intuitive interpretation of SLAs with respect to revenue generation.

[1]We do not use the term *entity* group either in order to avoid confusion between logical schema design and transaction design: For instance, we also plan to introduce *vertical sharding*, making a logical *entity* no longer an atomic unit of transaction scope.

The intuition is that, if the level of services changes, the amount that the provider can charge the client also changes according to the contract. Due to the limitations on the availability of infrastructure resources, the cloud service provider may not be able or choose to attend to all client requests at the highest possible service levels. Dropping/Increasing service levels cause loss/increase in the revenue. The loss of potential revenue corresponds to SLA function. Likewise, increasing the amount of infrastructure resources to increase service levels results in increased operational cost. As a result, the key problem for the provider is to come up with optimal service levels that will maximize its profits based on the agreed upon SLAs.

We note that the quantile-based SLAs are more commonly used in practice – especially for availability measures. If this is preferred, there exist techniques (e.g., [9]) that directly map quantile-based SLAs to per-query SLAs. However, based on our extensive interactions with numerous business organizations that provide services to real clients, they desire to be able to manage SLAs at the finest granularity level (i.e., per query) with multiple levels of delivery defined in the SLAs (i.e., step functions). The observation is that currently majority of the service providers only give availability SLAs to their clients represented in the quantile form but not other types of SLAs such as latency, throughput etc. Also, lack of formal models and tooling to enable finer granularity level SLA management is a major inhibitor for businesses to adopt various types of SLAs and also varying levels. Our research aims at advancing the state-of-the art in that area and helping service providers by working with them.

4.1 iCBS: Efficient Cost-Based Scheduling

In the cloud computing environment, service providers offer vast IT resources to large sets of customers with diverse service requirements. Obviously the service provider may derive different revenues from different clients as the clients may have varying SLA and price agreements. Then one of the key questions the service provider has to answer is: "How should I prioritize the queries in the system for execution in way that will maximize the profits?" Naturally, the answer to this question should consider the constraints, such as available resources, and other inputs, such as customer SLAs etc. The prioritization of queries can be defined as a query scheduling problem, which is our focus in efficient cost-based scheduling project.

The scheduling is a mature problem and has long been extensively studied in various areas such as compute networks, database systems, and Web services. There exist many different scheduling policies and there is a vast amount of theoretical results and practical analysis on various scheduling policies. However, in the majority of existing work on scheduling, the performance metrics are low-level performance metrics (e.g., average query response time or stretch). More recently, researchers and practitioners have paid more and more attention to metrics other than the system-level ones. One such new metric is the cost-based metric. For example, instead of optimizing query response time, the main target of a scheduling policy can be to reduce the total query cost, while for each query there is a certain mapping between its response time and the corresponding query cost. Such a cost-based metric is well suited for cloud computing where profit plays a central role and so we believe it is worth investigating the cost-based scheduling problems in cloud services. We mainly focus on two requirement areas while designing a scheduling algorithm for this purpose. First, we need a competitive computational complexity as we expect a cloud service delivery system has to manage very large query queues, where efficiency is crucial. Second, the scheduling algorithm should be capable of handling comprehensive set of SLA functions that are representative of typically business contact cases. With those requirements in mind, we developed an efficient cost-based scheduling algorithm, called *iCBS* [6] by building on the foundation of a previous work in this area [11, 12]. iCBS has the following main characteristics and advantages: *1)* It has a competitive time complexity of $O(\log^2 N)$ over other SLA-aware scheduling algorithms. The competitive complexity is achieved by using certain techniques in computational geometry thereby making the cost based scheduling feasible for query scheduling in the cloud-based systems. *2)* It can handle wide range of SLA function families.We study cost-based scheduling for a special family of cost functions; namely piecewise linear SLAs. Piecewise linear SLAs are easy to describe in natural language and so preferred for business contracts. We implemented and demonstrated that for many special types of piecewise linear SLAs, iCBS can achieve $O(\log N)$ time complexity. Other than capturing various rich semantics, piecewise linear SLAs also make many computations in cost-aware scheduling more tractable, which is a key to our work.

4.2 SmartSLA: Virtualized Resource Management for Cloud Databases

In a cloud computing environment, multi tenancy

is a common practice where resources are shared among different clients. SmartSLA[2] project [16] focuses on intelligently managing and allocating resources among various clients in a multi tenanted cloud database environment. SmartSLA consists of two main components: the system modeling module and the resource allocation decision module. The system modeling module uses machine learning techniques to learn a model that describes the potential profit margins for each client under different resource allocations. Based on the learned model, the resource allocation decision module dynamically adjusts the resource allocations in order to achieve the optimum profits.

More specifically the problem is that he service provider should intelligently allocate limited resources, such as CPU and memory, among competing clients. On the other hand, some other resources, although not strictly limited, have an associated cost. Database replication is such an example. It is known that adding additional database replicas not only involves direct cost (e.g., adding more nodes), but also has initiation cost (e.g., data migration) and maintenance cost (e.g., synchronization). We view the successful management of resources as follows:

Local Analysis : The first issue is to identify the right configuration of system resources (e.g., CPU, memory etc.) for a client to meet the SLAs while optimizing the revenue. Answers to such a question are not straightforward as they depend on many factors such as the current workload from the client, the client-specific SLAs, and the type of resources.

Global Analysis : The second issue that a service provider has to address is the decision on how to allocate resources among clients based on the current system status. For example, how much CPU shares or memory should be given to the gold clients versus the silver ones and when a new database replica should be started, etc.

In the SmartSLA system architecture the system modeling module mainly answers the Local Analysis questions, and the resource allocation decision module is responsible for the Global Analysis questions.

4.3 ActiveSLA: Profit-Oriented Admission Control

Compared to traditional database systems, the databases systems hosted in the cloud usually serve more diverse clients (e.g., through multi-tenancy) and therefore face more unpredictable workloads. Due to economic considerations, cloud database providers try to avoid resource overprovisioning while accounting for simultaneous peak workloads from a large number of clients. Consolidating multiple clients in shared infrastructures is a very commonly used approach by the cloud providers. Such a consolidation affords greater economies of scale and fixed cost distribution. However, managing the overloading becomes a much more crucial problem in such environments.

The admission control has been proposed to resolve the system overloading problem. With admission control, when the system is near an overloading condition, new queries are either throttled (e.g., [8]) or rejected (e.g., [15]) until the system condition improves. Although existing admission control techniques are helpful to alleviate system overloading, they do not work directly toward the main goal of cloud service providers—namely to maximize their profits by satisfying different SLAs.

We have designed and implemented an admission control framework, called ActiveSLA[3], for making prediction-based and profit-oriented admission control decisions, with a target of maximizing the expected profit of the database service providers [17]. The ActiveSLA framework is an end-to-end solution that consists of two main modules: a *prediction module* and a *decision module*. When a new query arrives, the query first enters the prediction module. The prediction module uses machine learning techniques and considers both the characteristics of the query and the current system conditions. The prediction module outputs the probability of the query meeting its deadline. The calculated probability and the query's SLA are sent to the decision module. The decision module decides either to admit the query or to reject the query up-front. Finally, the result of each admitted query is returned to the client and the actual execution time is piggybacked to the prediction module in real time. This piggybacked information can further help the prediction module to improve the accuracy of its future predictions by introducing new training data. Further details can be found in [17]. We believe the consideration of profit maximization in admission control presents new challenges that we address in our solution.

5. COSMOS: SEAMLESS MOBILITY BY CLOUDDB

The mobility of today is defined by the multitude of apps, which while working in isolation, can

[2]SmartSLA stands for "Resource Management for Resource-Sharing Clients based on Service Level Agreements".

[3]ActiveSLA stands for: Admission Control for Profit Improving under Service Level Agreements.

achieve a variety of tasks for the mobile user. The mobility of tomorrow is envisioned as one where mobile apps work together by sharing information to create a seamless mobile experience, where the focus is the mobility of the user but not the device.

In the COSMOS (stands for *Clouddb fOr Seamless MObile Services*) project, we are developing a multitenant, SLA-aware, cloud-based PaaS on top of CloudDB to provide the necessary support for *seamless mobility* [13]. The core component of COSMOS is the Sharing MIddLEware (SMILE), which provides the infrastructure for mobile apps residing on COSMOS to share data actively with one another. SMILE allows for management of SLAs on the shared data, which means that some serious technical challenges will have to be overcome in order to guarantee the desired level of access on the shared data to all those who access it. The key challenge is in ensuring that SLA guarantees are provided in the face of multiple users with diverse workloads and SLA requirements, while providing performance guarantees for the data owners in sharing data with others using performance isolation policies.

The ultimate goal of mobility is to ensure that the experience of a mobile user is a rich one. This means that individual apps (i.e., mobile services) should interact with the mobile user as if they exist in an ecosystem whose collective objective is to ensure that the mobile user can interact with the digital world in a *seamless* fashion. Mobile users frequently change their context as they navigate in their fast-paced daily lives. The desire is to ensure that mobile users stay connected to the digital world through mobile services as they move forth from one context to another – regardless of the kind of mobile device and sometimes even without a mobile device, which is largely irrelevant here. In this environment, the main constraints of mobility are the limited time, patience, and attentive span of the mobile user who is *on the go*. Although there have been recent efforts to significantly advance the capabilities of mobile devices to improve the user experience, more substantial improvements in user experience can be achieved if individual apps are cognizant to the limitations of the mobile user. In some sense, we want to move beyond traditional arguments that solely attribute the challenges of mobility to the limitations of the mobile device, but instead focus on how apps can provide a much better user experience. Apps that constantly adapt to the current context of the mobile users are said to exhibit "*seamless mobility*."

The seamless mobility can be achieved by apps working together to help the mobile user achieve tasks on the go. In that case the apps could greatly benefit from sharing information with one another. The mobility as we envision is far from what currently exists. Even though there is a rich variety of apps on mobile devices, they generally do not talk to one another or they try to do it an ad-hoc manner let alone share information to create a seamless mobile experience. The problem with most of todays connections among the mobile apps is that it is unidirectional and not scalable in the sense that every app needs to implement the API individually. In general, APIs are expensive to create and maintain, not to mention that they may not be expressive enough for most sharing needs between apps. Moreover, as apps are often hosted in the same cloud infrastructure or on different cloud infrastructures that can communicate in standard ways, there are other less expensive ways of enabling communications between apps hosted in the cloud.

COSMOS PaaS System: Mobile apps can be viewed as front ends driven from remote services, which are typically hosted on the cloud. For example, a weather app on a mobile device is essentially a front end that queries a data store on the cloud infrastructure for the weather conditions at a certain zip code. PaaS provides hosting, processing and querying of data for any mobile app that wishes to use its services. A PaaS is the most appropriate place to build a service for sharing as it typically hosts data from several other apps. Sharing between the apps can be provided as a service with little or no overhead to the apps that use it. Note that one critical aspect of sharing in this context that we believe would make sharing successful is the consistent nature in which mobile apps can refer to a mobile user using a small set of *key identifiers* (e.g., phone number, device id, simcard id). Sharing, in our context, adds value to all the parties involved by providing access to richer information on the mobile user.

COSMOS would be a PaaS offering for mobile apps built on NEC CloudDB [10], with the goal of supporting seamless mobility by enabling wide scale sharing between apps. We consider a RDBMS model of data storage, such that access to the hosted data is typically using SQL. In the PaaS setting, mobile apps that use the PaaS to host their databases are referred to as *tenants*. Usually a *PaaS provider* hosts several tenants in the same cloud infrastructure. In other words, the PaaS provider usually resorts to *multitenancy* for good resource usage and spreading of the operation cost among several tenants. To ensure that all the tenants get an acceptable level of service, in spite of sharing the infras-

tructure with several others, tenants negotiate SLA with the PaaS provider. SLA is a contract that describes the level of service a tenant requires on the data hosted with the PaaS. For example, an SLA could specify that the tenant would pay 10 cents for queries responded within 300ms, while the tenant would penalize the PaaS $1 if the execution time for the query exceeds 300ms. The PaaS provider, whose objective is to maximize profits, ensures that sufficient resources are available so that tenants do not miss their SLA deadlines too often as that results in a loss of revenue.

SMILE: Data Sharing in the Cloud: The key component of COSMOS is a middleware for sharing data. The Sharing MIddLEware, known as SMILE, enables sharing between a tenant t, who is the *owner* of the data and another tenant, referred to as a *consumer*, who wants access to t's data. Consider a scenario of two apps, say App-A and App-B, hosted on COSMOS that agree to share data. In particular, let us only consider the case of App-A, who is the data owner, agreeing to share data with App-B who is the consumer. For instance, App-A could be a calendar service, while App-B could be an airline ticket booking service that wants to query calendar appointments to determine if the user is traveling in the near future. If App-A wants to share some of its data with App-B, in a traditional scenario, App-A would create an API and share the details of the API with App-B. Now, App-B would use the API to access App-A's data. The advantage of this model is that App-A is *loosely coupled* with App-B in the sense that App-A is free to change its data layout without really affecting App-B as long as the API is suitably updated. However, App-A must setup the necessary infrastructure to create the API as well as keep updating it whenever its data layout changes. In our context, an API is an inefficient way to access App-A's data, especially if both App-A and App-B reside in the COSMOS system.

An extreme solution would be if App-A allows App-B to access its data directly. As App-A and App-B are both tenants in COSMOS, this can be trivially achieved. The drawback of this arrangement is that it leads to a tight coupling between App-A and App-B in the sense that if App-A changes its data layout, it has to coordinate with App-B. Other issues with sharing that are specific to a PaaS system like COSMOS are as follows. Imagine the scenario that App-B is a *hard-hitter* (i.e., issues queries at a high rate) of App-A's data, which would lead App-A to frequently miss SLA deadlines on its own data. Moreover, if App-A extensively shares its data with several other consumers, the access on the shared data may be exceedingly poor for all the parties involved without the PaaS investing additional resources to ensure that everyone gets reasonable access. Sharing may require substantial investment of resources from the COSMOS provider, where the COSMOS provider has setup a *materialized* shared space for App-B, which ensures that App-B's access on the shared space will not significantly affect App-A's queries. As materialized shared space takes up storage and is expensive to maintain so this solution, while attractive, must be used intelligently.

6. REFERENCES

[1] Apache HBase. `http://hbase.apache.org/`.

[2] CloudDB:A Data Store for All Sizes in the Cloud. `http://www.nec-labs.com/dm/CloudDBweb.pdf`.

[3] Google App Engine. `http://code.google.com/appengine/`.

[4] Project Voldemort. `http://project-voldemort.com/`.

[5] P. A. Bernstein, D. W. Shipman, and J. B. Rothnie, Jr. Concurrency control in a system for distributed databases (SDD-1). *ACM Trans. Database Syst.*, 5(1):18–51, 1980.

[6] Y. Chi, H. J. Moon, and H. Hacıgümüş. iCBS: Incremental cost-based scheduling under piecewise linear slas. *PVLDB*, 4(9), 2011.

[7] G. Eadon, E. I. Chong, S. Shankar, A. Raghavan, J. Srinivasan, and S. Das. Supporting table partitioning by reference in oracle. In *SIGMOD '08*, pages 1111–1122, 2008.

[8] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proc. of WWW*, 2004.

[9] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive quality of service management for enterprise services. *ACM Trans. Web*, 2008.

[10] H. Hacıgümüş, J. Tatemura, W. Hsiung, H. J. Moon, O. Po, A. Sawires, Y. Chi, and H. Jafarpour. CloudDB: One size fits all revived. In *IEEE World Congress on Services (SERVICES)*, 2010.

[11] J. M. Peha and F. A. Tobagi. A cost-based scheduling algorithm to support integrated services. In *INFOCOM*, 1991.

[12] J. M. Peha and F. A. Tobagi. Cost-based scheduling and dropping algorithms to support integrated services. *IEEE Transactions on Communications*, 44(2):192–202, 1996.

[13] J. Sankaranarayanan, H. Hacıgümüş, and J. Tatemura. COSMOS: A Platform for Seamless Mobile Services in the Cloud. In *IEEE MDM*, 2011.

[14] J. Tatemura and H. Hacıgümüş. Microsharding: A declarative approach to support elastic OLTP workloads. In *The 5th Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2011.

[15] S. Tozer, T. Brecht, and A. Aboulnaga. Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *Proc. of ICDE*, 2010.

[16] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacıgümüş. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In *ICDE*, 2011.

[17] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacıgümüş. ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers. In *SoCC*, 2011.

Integration of VectorWise with Ingres

Doug Inkster, Actian Corporation
Marcin Zukowski, Actian Netherlands B.V.
Peter Boncz, CWI

ABSTRACT

Actian Corporation recently entered into a cooperative relationship with VectorWise BV to integrate its VectorWise technology into the Ingres RDBMS server. The resulting commercial product has already achieved phenomenal performance results with the TPC-H industry standard benchmark, and has been well received in the analytical RDBMS market. This paper describes the integration of the VectorWise technology with Ingres, some of the design decisions made as part of the integration project, and the problems that had to be solved in the process.

1. INTRODUCTION

The Ingres project of the 1970's at UC Berkeley was one of the first attempts to practically apply the theory of the relational model to solving problems of data storage and access. The project led to much research and many published papers [6], some of which were fundamental to the early development of relational database systems. Berkeley Ingres subsequently formed the basis of numerous commercial ventures including the Ingres RDBMS introduced by Relational Technology Inc., formed by several members of the original Ingres research team in 1980. While its market share is not what it once was, Ingres has been continually developed since its original market introduction. After a succession of acquisitions and following its release into open source in 2004 by CA Technologies, Ingres was reintroduced as an independent RDBMS by the newly founded Ingres Corporation in 2005, and its development and promotion has been pursued even more aggressively since then. In 2008, Ingres Corporation (renamed as Actian Corporation in September 2011) entered into a commercial agreement with VectorWise BV, a spin-off company from the Dutch research institute CWI (Centrum voor Wiskunde en Informatica). VectorWise was created to bring to market the technology that was developed by the MonetDB/X100 project [1, 2, 10]. The goal of the agreement was to integrate the VectorWise technology with the Ingres database server to produce a viable commercial RDBMS that provided the accepted components of enterprise class database management together with the extreme performance of the VectorWise engine as demonstrated in the numerous research papers generated from the MonetDB/X100 project.

2. VECTORWISE

The MonetDB/X100 research project was initiated at CWI to extend the ideas behind its open source MonetDB column store database to workloads where the data set size is significantly larger than memory. This led to the *vectorized query execution* paradigm proposed in the Ph. D. thesis of Marcin Zukowskix100phd, that conserves and even improves the raw computational efficiency of MonetDB, but avoids its policy of full intermediate result materialization, hence improving scalability. The focus on large disk-resident workloads led to further innovations in highly efficient compression methods [10] and Cooperative Scans [9] to to reduce I/O bandwidth pressure. Like its ancestor MonetDB, X100 is a column store database server designed for read-mostly applications performing complex analytic queries on large volumes of data. That said, it also employs sophisticated updating mechanisms based on Positional Delta Trees (PDTs) that give it acceptable performance in update applications from moderate to heavy [2].

The design goals of X100 were to build upon the lessons learnt in the MonetDB project[1] and to exploit features of modern computer hardware architectures to deliver improved retrieval performance to relational queries. In addition to the benefit of reduced disk access inherent in column store databases, the X100 design takes a holistic view of disk, main memory, CPU cache and CPU to produce a database server that balances resource con-

[1]See `http://www.monetdb.org`

sumption from disk bandwidth, to memory bandwidth to CPU instruction cycles. Among the more novel ideas incorporated into the X100 architecture is a lightweight compression scheme which uses different techniques depending on the data type and value distribution. The compression algorithms are simple enough to avoid diverting CPU resources from actual query execution, yet effective enough to reduce the need for disk space and increase the effective bandwidth of disk transfer. Moreover, the data is left in its compressed form even in main memory. Only when it is ready to be used by the actual operators of the executing query plan is it decompressed. This technique allows better utilization of the scarce bandwidth between CPU cache and memory.

The primary source of the improved performance realized by the VectorWise engine is its server architecture. Data flows passed through algebraic X100 operators are kept in columnar form in an effort to reduce the execution overhead of row store database engines. Moreover, the columns are processed in *vectors* of a size designed to maximize the performance achieved in CPU cache. The server functions that implement the operators are written to exhibit strong locality of reference and to take advantage of such machine features as instruction pipelining and compiler generation of SIMD (single instruction, multiple data) instructions. The vectorizing of operators to use SIMD instructions allows operations to be executed on many data items simultaneously and greatly reduces the machine cycles consumed for each tuple processed by the query engine. During the process of integration with the Ingres server, the VectorWise engine has been enhanced with other features typically associated with high performance analytical query execution. Parallel execution of complex queries using the Volcano model, revamped sorting, disk overflow hashing and many more scalar functions are just some of the improvements that have been made to VectorWise during the integration process.

The interface to the VectorWise engine is a low level, relational algebra language. Operators are defined for session management, DDL and DML (samples shown in Table 1). It is easily compiled into the code form used for execution and depends on queries being already optimized to get the best performance. Though the VectorWise compilation process does incorporate a rewriter to perform localized optimization, the generation of optimal queries is primarily the responsibility of the coder of the query. Figure 1 shows a simple example of an SQL query and the corresponding VectorWise algebra

Operators Producing a Flow of Tuples
Project(Flow input, Expr* calc)
Select(Flow input, Expr cond)
Sort(Flow input, Expr* orderby)
TopN(Flow input, Expr* orderby, int N)
Aggr(Flow input, Expr* groupby, Expr* aggregates)
HashJoin1(Flow in1, Expr* k1, Flow in2, Expr* k2)
HashJoin01(Flow in1, Expr* k1, Flow in2, Expr* k2)
HashJoinN(Flow in1, Expr* k1, Flow in2, Expr* k2)
MergeJoin1N(Flow in1,Expr* k1,Flow in2,Expr* k2)
HashSemiJoin(Flow in1,Expr* k1,Flow in2,Expr* k2, Expr cond
HashRevSemiJoin(Flow in1,Expr* k1,Flow in2,Expr* k2, Expr cond
HashAntiJoin(Flow in1,Expr* k1,Flow in2,Expr* k2, Expr cond
HashRevAntiJoin(Flow in1,Expr* k1,Flow in2,Expr* k2, Expr cond
DDL and DML Operators
CreateTable(str table, str* colspec)
Insert(str table, Flow input)
Delete(str table, Flow input, Expr* key)
Modify(str table, Flow input, Expr* key)
Start()
Commit()
Abort()

Table 1: Sample X100 Algebra Operators

query. The `MScan` operator creates a data flow containing the `sno` and `qty` columns from the `sp` base table. `MScan` stands for Merge-Scan, as it scans columnar data from disk while merging in possible committed updates present in the PDTs [2]. `Aggr` performs a hash aggregation grouping the input flow on `sno` and computing the sum of `qty` values for each group. The `Project` operator in VectorWise is used to compute new columns using input columns, functions and simple expressions – but it does not eliminate duplicates in the manner of pure relational projection. In this example, the `Project` just selects the two relevant output columns from the aggregation flow.

3. INTEGRATION WITH INGRES

The first question asked about this project was whether it was even feasible to integrate the type of leading edge database technology as embodied in VectorWise with a mature, row-based RDBMS server such as Ingres. The VectorWise server, regardless of its implementation, still provides a relational interface to the users of the data it maintains. Such is the continuing strength of the relational model that, even though user interface languages

SQL:

```
SELECT sno, SUM(qty) FROM sp GROUP BY sno;
```

VectorWise:

```
Project(
    Aggr(
        MScan('sp', ['sno', 'qty']
        ), [sp.sno], [col2 = sum(sp.qty)]
    ), [sno, col2]
)
```

Figure 1: Simple SQL query in X100 Algebra

may differ, access to contained data is achieved with analogous mechanisms. Different relational languages can almost always be compiled into compatible database interfaces. So queries, both DDL and DML, can be coded in SQL, given to the Ingres server to be processed as necessary, and then passed seamlessly by Ingres to the VectorWise engine for execution. In fact, the original Quel language of Berkeley and commercial Ingres can also be used for expressing queries on VectorWise databases.

3.1 Adding VectorWise Tables to Ingres

Two primary mechanisms were available for defining VectorWise tables to the Ingres server. Ingres has long included a gateway capability that allows tables defined in federated RDBMS', and even flat file systems, to be "registered" in the Ingres information schema. Heterogeneous queries on such tables are compiled in Ingres, then passed to the database or file servers in question for processing. Wrapper functions are required to be written for each such interface to allow exchange of definitions, data and transaction processing commands. To develop such an interface to allow access to data stored in a VectorWise database would have been a natural use of the gateway facility, one which would have been quite straightforward to implement.

The ease of defining a gateway interface from Ingres to VectorWise was offset, however, by several factors. Possibly the very first design goal of the entire integration project was to avoid burdening the VectorWise processing technology with the overheads of the Ingres row store architecture as much as possible. The gateway approach delivers rows back to the Ingres server one at a time to be returned to the user and this was deemed an unacceptable bottleneck. Moreover, for a better user experience, it was determined that the interface from Ingres to VectorWise should be as seamless as possible and putting VectorWise tables in the same category as tables created in other RDBMS' would impede our ability to provide, among other things, an integrated transactional model for updates.

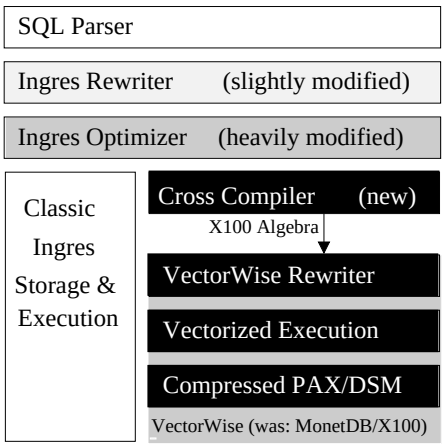


Figure 2: Architecture of Ingres VectorWise.

The decision to fully integrate VectorWise tables was consistent with the fact that Ingres already supported several table storage options, specified when tables are created. Rows of "HEAP" tables are simply stored sequentially as they arrive in the database, kept in a storage heap that extends automatically as rows are added. "ISAM" and "BTREE" are indexed sequential options and "HASH" stores rows scatters in the storage space, based on a hash of their key column values. VectorWise tables are simply defined by a new Ingres table storage option using the "structure=" parameter of the Ingres `CREATE TABLE` statement, analogous to the HEAP, ISAM, BTREE and HASH options already supported.

Various Ingres DDL statements have been extended to support the definition of VectorWise tables and indexes. Definitions of VectorWise tables and columns are recorded first in the Ingres information schema so that Ingres can recognize references to VectorWise tables and compile queries accordingly. The DDL is then passed to VectorWise so that definitions can also be recorded in local VectorWise catalogs for VectorWise engine use. Most importantly though, from the user's perspective the same interface is used to define and maintain VectorWise tables as those of native Ingres. Figure 2 depicts the final architecture of the integrated system, in which dark boxes correspond to (heavily) modified components. In the following, we discuss these modifications in more depth.

3.2 Optimizing VectorWise Queries

While the notion of integrating different table organizations into an existing database server is quite reasonable, generating optimal query plans for table structures as radically different from classic row

stores as the VectorWise column store takes a bit more of a leap of faith. However, arguably the most critical function of query optimization is the estimation of base table cardinalities in the presence of restriction predicates and intermediate result cardinalities in the presence of joins and aggregations. The remaining activities of a query optimizer are the enumeration of all possible, or at least reasonable, query plans and the association of cost estimates with each. These are primarily mechanical processes that can easily be extended to accommodate new data organizations.

Ingres was the first commercial RDBMS to use histograms as a method for representing value distributions of the data in a column, histograms being but one of numerous novel techniques introduced to the Ingres optimizer from the Ph. D. research of Robert Kooi[4]. As such, estimations of predicate selectivity and resulting cardinality have always been strengths of its query optimizer. Moreover, Ingres histograms are built by a utility that simply executes SQL queries on the underlying data. Row counts for each distinct value are accumulated in the utility itself, or by the Ingres server using aggregate queries, depending on the ratio of distinct values to rows in a table. Basic SQL retrieval support for VectorWise data allowed histograms to be built for VectorWise tables very early in the integration process and provided a basis for using the existing Ingres optimizer to generate acceptable query plans for VectorWise queries.

Therefore, the initial creation of query plans by Ingres for use in the VectorWise engine simply used the same cost algorithms as are used for queries on native Ingres tables. Certain Ingres-only join techniques were removed from the set of target strategies, but the remaining operations such as hash or merge joins were compiled identically for VectorWise and native Ingres queries. The resulting plans were quite promising, though certain characteristics of the VectorWise execution model definitely required additional consideration. Some of the changes to address these characteristics are described later in Section 3.2.2.

3.2.1 Changes to the Ingres Rewriter

Several categories of change were implemented in the rewriter component of the Ingres query optimizer. Not all functions supported by the Ingres dialect of SQL are supported in syntax by the VectorWise engine. Examples include statistical aggregate functions such as `stddev_pop`, `stddev_samp`, `var_pop`, `var_samp`, `corr`, `regr_slope`, `regr_intercept`, and so forth. These functions all have well known expansions into more basic operators such as multiplication, division, exponentiation, etc. and are defined that way in the SQL standard[3]. So the parse tree fragments constructed by the Ingres parser for statistical aggregates are detected and rewritten by the optimizer rewriter as more primitive sequences of functions and operators that are supported by the VectorWise engine.

Like all commercial implementations of SQL, the Ingres RDBMS rewrites many forms of `WHERE` clause subqueries into flattened joins with the containing queries. This allows more efficient execution strategies to be developed. However, some forms of subquery are very difficult to flatten and native Ingres relies on a nested loop "subquery" join operator to handle such cases. For each row of the containing query, the subquery is effectively executed with the current set of correlation variables (with some optimizations to avoid continual reevaluation), to determine the truth value of the subquery predicate. Subqueries that are connected to a `WHERE` clause by an `OR` are examples of the Ingres use of the subquery join, rather than some complex flattening strategy.

The VectorWise engine does not support any nested loop join operators. This has required the development of additional novel flattening strategies in the Ingres rewriter to handle subqueries in VectorWise queries that were not flattened for classic Ingres. These new subquery flattening strategies are generic in that they are also applicable to native Ingres queries and will deliver performance benefits to existing users of Ingres. There are numerous such examples of the symbiosis between the VectorWise integration project and native Ingres.

Figure 3 shows a `>= ALL` subquery and how it can be flattened to compute the `MAX` and `COUNT` of the comparand value, as well as `COUNT(*)`, then restricts the containing query to rows for which the subquery has either no result rows or result rows that all contain `null`, and for which either all subquery rows are `null` or the maximum subquery value is less than or equal the containing row comparand value. VectorWise has the ability to re-use the results of any operation (joins, projections, etc.) to be reintroduced later in the execution of the same query, such that proper graph-shaped query plans are possible. The Ingres rewriter already has the ability to identify some common table-level expressions so that their results can be computed once, and cached (typically in a lightweight temporary table) for reuse in the same query. This technique is used, for example, when the same aggregate view is referenced more than once in a single query. For

SQL:

```
SELECT * FROM p WHERE pno >=
 ALL (SELECT pno FROM sp WHERE QTY = 100)
```

VectorWise:

```
Project (
 Select (
  Select (
   CartProd(
    MScan (
     'p', [ 'pno', 'pname', 'color', 'weight', 'city']
    ) [ 'est_card' = '6' ] ,
   Aggr (
    Select (
     MScan (
      'sp', [ 'qty', 'pno']
     ) [ 'est_card' = '12' ] , ==(sp.qty, sint('100'))
    )  [ 'est_card' = '1' ] ,
    [] ,
    [TRSDM_3 = max(sp.pno),_TRSDM_4 = count(*),
     jTRSDM_5 = count(sp.pno)] , 200
    ), 1
   ), || (==(TRSDM_4,TRSDM_5), ==(TRSDM_4, sint('0')))
  ), || (==(TRSDM_4, sint('0')), >=(p.pno,TRSDM_3))
 )  [ 'est_card' = '3' ] ,
 [p.pno, p.pname, p.color, p.weight, p.city])
```

Figure 3: Flattened <compop> ALL subquery.

VectorWise queries, however, the Ingres rewriter has been extended to perform a more general search for common expressions. Resulting query fragments are then executed once with their results cached by the VectorWise engine. The cached result may then be referenced elsewhere in the same query without the need to rematerialize the rows from scratch.

3.2.2 *Optimizer Changes*

While the Ingres query optimizer generated acceptable plans for VectorWise queries almost immediately, there are certain capabilities that are specific to the VectorWise engine that required additional properties to be tracked during query plan enumeration. Ingres has long supported referential and primary key/unique constraints; however its optimizer has never exploited the presence of constraints while compiling queries. Accurate histogram-based cardinality estimates of restrictions and joins largely superseded the need to consider the implications of join predicates that map to referential relationships between tables in a query. In contrast, VectorWise has several features that can only be targeted with knowledge of the constraints that underlie the tables referenced in a query. As a result, a new information schema table was defined in Ingres to define more compactly the mapping of columns in a referential relationship. Rows are added to this table during the execution of the DDL statements that define referential constraints. Corresponding code was introduced to the Ingres optimizer to identify join predicates in a query that map to referential relationships.

The first benefit of such information is that the optimizer can more easily identify and estimate costs for joins that can take advantage of the join indexes that VectorWise will optionally create to support referential relationships. Merge joins between a join index in a referencing table and the tuple ID of the corresponding referenced table are the most efficient joins that VectorWise can perform.

Referential relationships and unique/primary key constraints are also the basis (along with the grouping columns of intermediate aggregate results) of the functional dependency property tracked for each node of a query plan for a VectorWise query. Functional dependencies allow the leveraging of a feature of the VectorWise aggregation operators. Those operators include an operand list describing the aggregate results to be computed and an operand list describing the grouping columns upon which the aggregation is to be performed. Columns in the grouping list whose values are functionally dependent on one or more other columns in the list do not logically contribute to the determination of distinct groups over which the aggregation is performed. As such, VectorWise allows them to be designated with the `DERIVED` attribute, in which case they are left out of the grouping operation. The requirement to include all columns of an SQL select-list in either an aggregate function or the group by clause of the query leads to many queries that include large text columns in grouping lists, simply to allow them to be included in the result row. Use of the `DERIVED` attribute allows such columns in the grouping column list without imposing a potentially significant overhead on the hashing and comparison operations used to group the data. Figure 4 shows an aggregation on `sno` and `sname`. By virtue of the unique constraint on `sno`, `sname` is functionally dependent on it and can be flagged as `DERIVED` in the aggregation, not needing to be part of the actual grouping operation taking place. Like most relational query optimizers, the Ingres optimizer tracks ordering properties of nodes of a query plan. This allows it to eliminate sorts at various points in the query plan and to use the much more efficient ordered aggregation operator to perform grouping operations, rather than relying on hash techniques. However, ordered grouping only requires its input to be clustered on the grouping column values. That is, all of the rows with the same values in the grouping

SQL:

```
  SELECT s.sno, sname, SUM(qty)
    FROM s, sp
   WHERE s.sno = sp.sno
GROUP BY s.sno, sname;
```

VectorWise:

```
Project (
 Aggr (
  HashJoin01 (
   MScan (
    'sp', [ 'sno', 'qty']
   ) [ 'est_card' = '12' ] , [ sp.sno ],
   MScan (
    's', [ 'sno', 'sname']
   ) [ 'est_card' = '5' ]  , [ s.sno ]
  ) [ 'est_card' = '5' ] ,
  [s.sname DERIVED, s.sno] ,
  [col3_3 = sum(sp.qty)
 ) [sno_1 = s.sno, sname_2 = sname, col3_3]
)
```

Figure 4: DERIVED designator in aggregation

columns must be contiguous. Clustering is a weaker property than ordering, and the Ingres optimizer has been extended to track it in addition to ordering to enable the identification of grouping operations that can be performed with the more efficient VectorWise `OrdAggr` operator, rather than the hash-based `Aggr` operator.

3.3 Query Plan Cross Compiler

The key component in the integration of Ingres and VectorWise is a cross compiler which generates a VectorWise query from an optimized Ingres query plan. Ordinarily, the Ingres query compiler transforms the optimized query plan into a code form executable by the Ingres query execution engine. For a query on a VectorWise database, the Ingres code generator invokes the cross compiler to transform the optimized plan into query syntax to be executed by the VectorWise engine. The cross compiler traverses the optimized plan using recursive descent, much like the code generator does for native Ingres code generation. But as the plan tree is descended, syntax is emitted for each of the corresponding operators of the VectorWise relational algebra, and on return from the descent the columns, expressions, literals and other parameters of the operators are emitted. The result is a query which builds data flows from base table access and successively refines them through the execution of the nested operators such as joins, aggregations and unions.

One source of difficulty within the cross compiler is the fact that the VectorWise query references database entities (tables and columns) by name, whereas Ingres query plans compile into buffer references and offsets. In fact, entity names are effectively removed from the query when it is turned into a parse tree – before query optimization takes place. The cross compiler goes to great lengths to avoid name ambiguities and scope errors by generating distinct table and column names and making heavy use of table name qualifiers in column references. Different instances of the same table and column names are modified to assure distinctness, sometimes at the expense of readability of the resulting syntax. A simple exception table is used to identify Ingres expression operators that do not have an exact equivalent in the VectorWise algebra. For example, VectorWise supports `is_null` and `like` operators, but not the negated `is_not_null` or `not_like` operators. The exception table has entries for the Ingres `is not null` and `not like` operators that result in the `!` (not) operator being prepended to the supported VectorWise `is null` and `like` operators.

There are also numerous examples of implementation effort being divided between the VectorWise engine and the cross compiler. VectorWise does not have native support for windowed functions such as `rank()`, `dense_rank()`, `ntile()` and so forth. To relieve the VectorWise development team of the entire burden of implementing these complex functions, the cross compiler took on the responsibility of generating syntax to perform the partitioning and ordering, as well as the joining together of results of window functions using differing window specifications. The VectorWise engine supplies more primitive functions to first partition the input rows, then to identify peer groups within each partition based on the requested window ordering. The results of those functions are then used to compute the various windowed functions supported by Ingres/VectorWise.

Example 5 shows a query that computes the `rank()` function over two distinct window specifications. The cross compiler computes the first rank value and a sequentially assigned row number for each row in the first pass of the result set. This initial result is cached using the VectorWise intermediate result re-use mechanisms described in Section 3.2.1, which surfaces here in the assignment into the `IIWINSQNAME1` identifier of a subquery result, and the subsequent re-use of this identifier. The cached result is re-sorted to generate the second rank value and the results of the two computations are joined back together on the computed row number.

SQL:

```
SELECT sno, pno,
   RANK() OVER (PARTITION BY sno ORDER BY qty)
        AS srank,
   RANK() OVER (PARTITION BY pno ORDER BY qty)
        AS prank
FROM sp;
```

VectorWise:

```
Project (
   HashJoin01(
       IIWINSQNAME1 =
      Project (
         Sort (
            MScan (
                'sp', [ 'qty', 'sno', 'pno']
            ) [ 'est_card' = '12' ] , [sp.pno,
                          sp.qty]
        ), [TRSDM_0 = diff(sp.pno),
          TRSDM_1 = rediff(TRSDM_0,sp.qty),
          IIWINRNUM10  = rowid(uidx('0')),
          sp.qty, sp.sno, sp.pno, prank =
                  sqlrank(TRSDM_0,TRSDM_1)]
      ), [IIWINRNUM10],
      Project(
         Sort(
            Project(
               IIWINSQNAME1, [IIWINRNUM11 =
                   IIWINRNUM10, sp.sno, sp.qty]
            ), [sp.sno, sp.qty]
         ), [TRSDM_3 = diff(sp.sno),
           TRSDM_4 = rediff(TRSDM_3,sp.qty),
           srank =sqlrank(TRSDM_3,TRSDM_4),
           IIWINRNUM11]
      ), [IIWINRNUM11]
   ), [sp.sno, sp.pno, srank, prank]
)
```

Figure 5: Windowed functions with differing window specifications

3.4 VectorWise Rewriter

The X100 Algebra plans created by the cross compiler lack certain details that were felt not relevant during the (join-order) query optimization phase. These typically concern highly VectorWise-specific features. For instance, VectorWise represents decimal and date/time data types using simple integers of different widths (1, 2, 4, 8 and 16 bytes) that are carefully selected to be of the minimal width needed to prevent overflow. This selection is based on min/max statistics kept for all columns. This mapping of logical SQL data type to physical type is performed by a VectorWise rewriter that operates on the already optimized algebraic query plan, as a post-processing step.

Similarly, VectorWise represents `null`-able columns as a pair of a value- and boolean-column, where the boolean column indicates whether the tuple value `is_null`. This representation and resulting processing model keeps functions null-oblivious and avoids the need for null checks deep inside the execution primitives. This absence of checks in turn conserves the efficiency of the vectorized model, maximizing e.g. SIMD opportunities. The extra boolean `null` columns, and the propagation of these, are also handled by a phase in the VectorWise rewriter.

For this new VectorWise rewriter we used a pattern-matching tool called Tom[2] that can be embedded in C/C++ code to aid with the formulation of rewrite rules. The presence of a separate, column-oriented rewriter has aided the project as certain functionality can be engineered with much less impact to the main Ingres optimizer. Besides the mentioned physical typing and null optimizations, there are various other rules implemented and many opportunities for new rules. Notably, this column-oriented rewriter was also used to implement multi-core query parallelization. Thus, parallelism is planned posterior to main join-order query optimization. The same approach is taken by many other systems, but may miss possibilities an integrated optimization method could potentially spot. Thus, the flexibility of having a post-processing VectorWise rewriter also leads to certain design trade-offs of the scope of query optimization versus the complexity of development.

3.5 Query Execution

The VectorWise server runs in a separate process from the Ingres server. It is initiated by a VectorWise interface layer that exists in the Ingres server. This does not occur until the first user request to Ingres is received that actually requires access to the VectorWise server. Execution by Ingres of queries destined for the VectorWise server is a relatively straightforward process, primarily following the paradigm of making the interface as simple as possible to take full advantage of the performance of the VectorWise engine. As mentioned in Section 3.1, DDL statements that affect the Ingres information schema are executed first in Ingres to perform necessary information schema changes, then passed to the VectorWise interface where they are converted to X100 algebra for delivery to the VectorWise engine where an analogous process records information in the VectorWise catalog. Transaction synchronization is performed to assure that errors in either the Ingres server or the VectorWise server result in rollback of the effects of the statement in both servers.

Queries are optimized in Ingres and transformed to X100 syntax by the cross compiler as described in Section 3.3. The syntax is then delivered to the

[2]See `http://tom.loria.fr`

VectorWise interface, along with the address of a vector of row buffers to expedite the return of retrieved result rows from the VectorWise server to the user. The size of the buffer array is configurable. As rows are returned to the VectorWise interface in the Ingres server, they are reformatted to match the data types expected by the user and contained in a tuple descriptor that is passed to the interface along with the query.

`UPDATE` and `DELETE` statements are executed in the same manner, though `INSERT` statements go through the usual Ingres insert processing and are diverted to the VectorWise interface before the row is written. A corresponding VectorWise `append` operation is prepared there and delivered to the VectorWise engine for execution. `INSERT .. SELECT` and `CREATE TABLE .. AS SELECT` statements differ in that the retrievals that provide the row images to be inserted may be executed on native Ingres tables or on VectorWise tables. This allows a path for converting Ingres tables to VectorWise tables for existing users. When the `SELECT` references Ingres tables, row images are built from the result rows of the retrieval in the Ingres query executor and delivered in arrays to a bulk load interface to the VectorWise engine, analogous to that used for the Ingres `COPY` statement. If the `SELECT` only references VectorWise tables, a VectorWise statement is composed by the cross compiler to stream the result of the retrieval into a VectorWise `append` operator that stores the newly formatted rows. Session and transaction management details are exchanged between the servers to assure recovery from any type of statement failure.

4. FUTURE ENHANCEMENTS

The short term goals of Ingres/VectorWise include functionality enhancement and further integration with Ingres and native Ingres tables. Currently, queries may reference all VectorWise or all native Ingres tables, but not a combination of the two (with the exception of `INSERT .. SELECT` and `CREATE TABLE .. AS SELECT` statements as described in Section 3.5). For the benefit of existing users of Ingres, it is desired to permit heterogeneous queries that combine results from both native Ingres and VectorWise tables. The Ingres query execution model certainly permits such sophisticated query plans, but a significant amount of work is required both to optimize heterogeneous plans and to handle appropriate interfacing between the two engines as the queries are executed.

Another future project will be the enhancement of the Ingres query optimizer to include more specific knowledge of VectorWise cost parameters in the building of query plans. The strategy in the initial release of the product has been to track certain VectorWise-specific properties during the building of a query plan and take advantage of them when generating the corresponding VectorWise query. This enhancement will associate costs with the properties being tracked so that the presence of such properties can influence the selection of query plans.

Column store engines offer an opportunity to implement finer granularity query optimization. Traditional query optimizers focus on strategies for table access and joining. All columns from a given table that are required to solve a query are typically retrieved once and carried through the remainder of the execution of the query. Column stores offer the opportunity to defer retrieval of some columns from a table until the evaluation of restrictions on other columns from the same table have significantly reduced the cardinality of intermediate results. Longer text columns may then be retrieved later in query execution and joined to the intermediate results based on their ordinal positions in the table. This type of optimization is under consideration for both the Ingres optimizer and the VectorWise rewriter.

The VectorWise team maintains a working relationship with CWI, including an active intern program. Research projects include cooperative scans, advanced compression techniques and "just in time" compilation [5]. Because of its academic roots, the VectorWise/X100 project is widely known in the research community. Numerous university research facilities in Europe and North America have entered the academic licensing program.

A variety of projects are underway at several of these schools, including fine-grained multi-table clustering structures and XML storage using the Pathfinder [7] XQuery compiler. An obvious benefit to Ingres is the leveraging of this research into future enhancements to the Ingres/VectorWise commercial product.

5. SUMMARY

Considerable risk was involved in attempting the integration of leading edge technology such as that embodied by the VectorWise engine with a mature RDBMS such as Ingres, much of whose architecture was developed over 15 years ago. In particular, the mixing of column store concepts with row store concepts was not an intuitive thing to do. Due largely to the robustness of the relational model, this integration project has seen considerable success already. The integrated product has been publicly available since July 2010 and already has been well

accepted by the user community. In the first half of 2011 TPC-H results where published for VectorWise on the 100GB, 300GB and 1TB sizes, all using single servers. The 100GB result (QphH 251,561.7), is quite comparable with a Microsoft SQL Server 2008 R2 Enterprise Edition result (QphH 73,974) as both tests where performed on a dual-socket HP DL380 server with 144GB RAM and in total 12 Intel Xeon X5680 cores. In addition to the record breaking results for non-clustered systems in these TPC-H benchmarks, Ingres/VectorWise has been certified by numerous application vendors, with more on the way.

6. ACKNOWLEDGEMENTS

We would like to recognize the contributions of the late Bob Kooi to the commercial Ingres optimizer, many of which survive in Ingres to this day.

7. REFERENCES

[1] Peter Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-pipelining query execution. In *CIDR*, 2005.

[2] S. Héman, M. Zukowski, N.J. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *Proceedings of SIGMOD*, pages 543–554. ACM, 2010.

[3] International Standards Organization. *(ANSI/ISO/IEC) 9075-2,* SQL *Foundation*, 2008.

[4] R. Kooi. *The Optimization of Queries in Relational Database Systems*. PhD thesis, Ph. D. Thesis, Case Western Reserve University, 1980.

[5] J. Sompolski, M. Zukowski, and P. Boncz. Vectorization vs. compilation in query execution. In *Proceedings of the Seventh International Workshop on Data Management on New Hardware*, pages 33–40. ACM, 2011.

[6] M. Stonebraker. *The INGRES Papers: Anatomy of a Relational Database System*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[7] J. Teubner et al. Pathfinder: Xquery compilation techniques for relational database targets. *Technical University of Munich, Munich, PhD thesis*, 2006.

[8] M. Zukowski. Balancing vectorized query execution with bandwidth-optimized storage. 2009.

[9] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative scans: Dynamic bandwidth sharing in a dbms. In *Proceedings of VLDB*, pages 723–734. VLDB Endowment, 2007.

[10] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-Scalar RAM-CPU Cache Compression. In *ICDE*, 2006.

30 Years of PODS in Facts and Figures*

Tom J. Ameloot† Maarten Marx ‡ Wim Martens § Frank Neven ¶
Justin Van Wees ‖

1 Introduction

The present paper discusses some facts and figures representing 30 years of history of the ACM Symposium on Principles of Database Systems (PODS). The analysis was done for the occasion of the PODS Anniversary Event held in Athens on June 12, 2011. All data corresponds to the period from 1982 up to and including 2011 unless explicitly mentioned otherwise and treats invited papers as regular papers. As the PODS Pages [1] are actively maintained, we do not repeat information here which can easily be found there. Also, supplementary material including word clouds, a snap shot of the PODS co-author graph, harvested data, spreadsheets, scripts, and the blueprint for the original PODS 30 year anniversary T-shirt can be found there as well.

2 Papers and People

Over 30 years 961 papers were published at PODS by 990 distinct authors. Table 1 depicts a more detailed overview of the distribution of papers per author. A large majority of PODS authors, that is, 64%, only published once while 90% of the authors published at most four papers. Still, there is a core of almost 100 hardliners that published 5 or more papers. Some of them are real devotees. Their papers show up at PODS every year. Record holder is Leonid Libkin whose longest uninterupted period of consecutive publications spans a period of 14 years. He is followed at a safe distance by by Moshe Y. Vardi, Dan Suciu, Georg Gottlob, Jan Van den Bussche, and Ron Fagin who all published PODS papers in 9 consecutive years. Figure 1 gives an overview of the longest streaks. Similarly, one can consider the longest period of absence at PODS, that is, the maximal time period between two PODS publications in which no article was published at PODS. Figure 2 provides a histogram displaying the longest period of absence. It is interesting to point out that although Ron Fagin published papers in PODS for 9 successive years, he was also absent for 12 years. The overall record here is 18 years of absence. Word clouds visualizing the devotees and the absentees can be found on the PODS Pages. In Figure 3, we display authors by their PODS score where each n-author paper attributes a score of $\frac{1}{n}$ to each co-author.

Number of papers	Percentage of authors
1	64%
2	17%
3	5%
4	4%
5 or more	10%

Table 1: Distribution of the number of papers per author.

3 Co-Authors

Figure 4 displays the distribution of the number of authors per paper. Surprisingly, about 25% of the papers are single-authored, while about 80% of all papers have three authors or less. Figure 5 shows the evolution of the number of single-author papers, displaying a decreasing trend. The latter is in line with the observation that over time the average number of authors of a PODS paper increased from 1.9 to 2.8. The maximum number of authors on a paper is 8. The paper was presented at PODS 2009 as an invited talk entitled "A web of concepts". The reg-

*We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

†Hasselt University & Transnational University of Limburg; Diepenbeek, Belgium. Email: tom.ameloot@uhasselt.be. PhD Fellow of the Fund for Scientific Research, Flanders (FWO).

‡Informatics Institute, Universiteit van Amsterdam; Amsterdam, The Netherlands. Email: maartenmarx@uva.nl

§Technical University of Dortmund; Dortmund, Germany. Email: wim.martens@udo.edu

¶Hasselt University & Transnational University of Limburg; Diepenbeek, Belgium. Email: frank.neven@uhasselt.be

‖Informatics Institute, Universiteit van Amsterdam; Amsterdam, The Netherlands. Email: justin@vwees.net

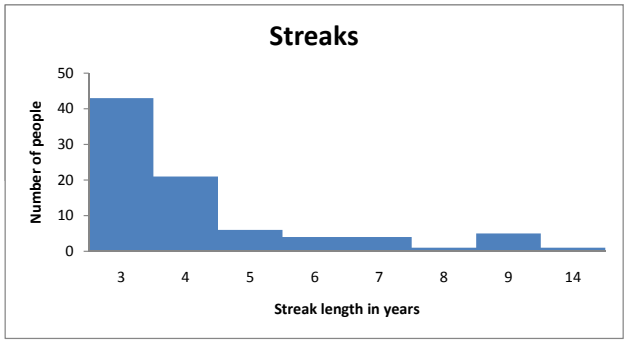


Figure 1: Histogram of the longest streaks (uninterrupted period of publishing by one author) at PODS.

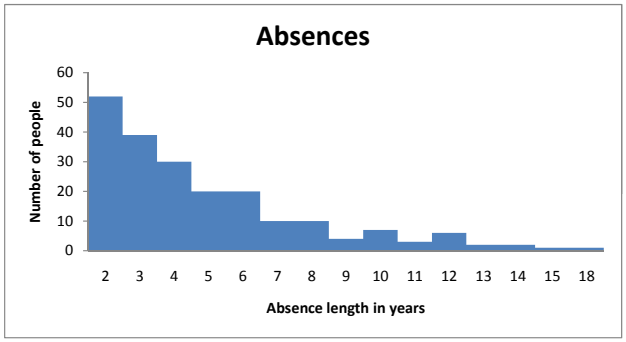


Figure 2: Histogram of the longest time periods between two successive PODS papers.

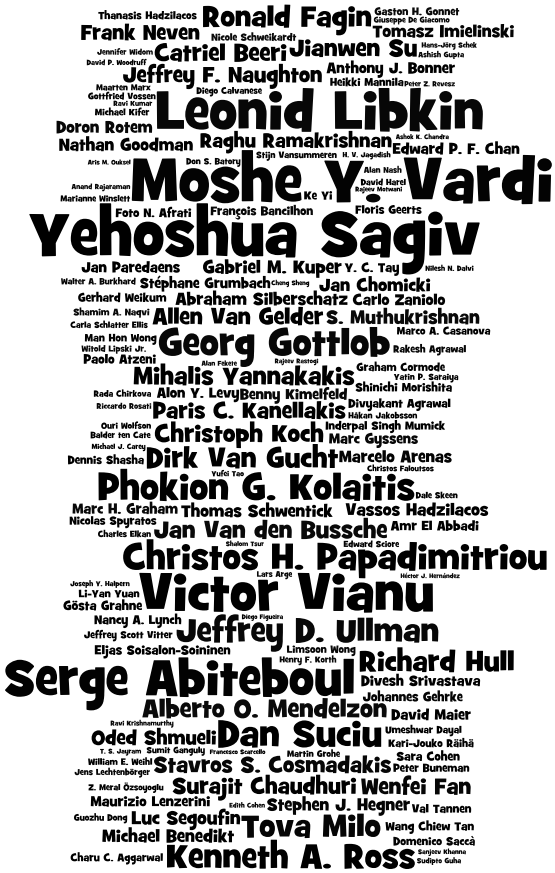


Figure 3: Most podsy researchers.

ular paper with most authors (7) was presented at PODS 2006 and was entitled "Achieving anonymity via clustering".

4 Community

Next, we try to provide insight into the community through an analysis of the structure of the co-author graph. We will use a number of measures from social network theory [2].

4.1 Betweenness centrality

Key nodes in a traffic network are those who lie on many shortest paths. The betweenness centrality of a node is the percentage of all shortest paths in a network which pass through that node. For each PODS year, we calculated this measure on the *complete* DBLP co-author graph restricted to the PODS authors, and restricted to publications up until that year. Thus, two PODS authors are connected in year x if they have a joint publication in DBLP (thus also in venues besides PODS) before or in year x. Referring to the most central person as the winner, Table 2 lists the most central persons per year and the number of times they won. The value of the centrality score of the winner has decreased over time. For instance, in 1982, 24% of all shortest paths went through Papadimitriou. The PODS co-author graph consisted then of 58 nodes. About ten years later, in 1991, Jeff Ullman won with 11%, while the winner from 2001 onwards, Serge Abiteboul, remained stable at 6% for 10 years and climbed to 7% in 2011. Table 3 lists the top 10 most central researchers in 2011 together with their score. The top 10 lists for all 30 years can be found at the PODS pages.

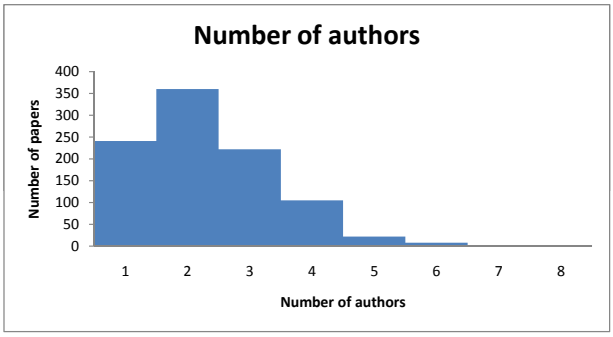


Figure 4: Distribution of the number of authors per paper.

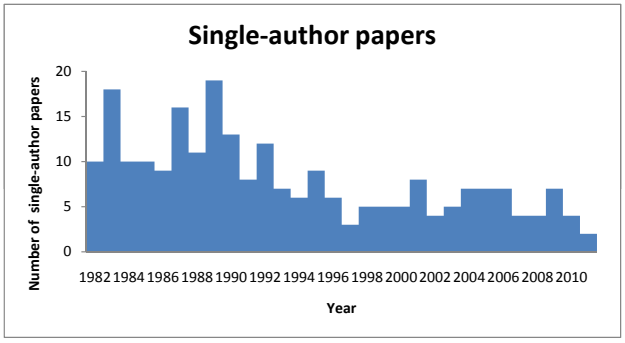


Figure 5: Evolution of single-author papers.

4.2 Connectedness

How well-connected is the PODS co-author graph, when restricting to only PODS publications? For starters, the PODS graph contains 990 nodes. The largest component grows steadily with an average of 26 new nodes yearly (over the last 20 years) and reaches 64% of all nodes in 2011 (7% in 1982, 36% in 1991 and 57% in 2000). Thus almost two thirds of all PODS authors are connected through a chain of PODS co-authorships. Now, what is the probability that two of your co-authors are co-authors themselves (a network transitivity measure)? This goes from 68% in the first PODS year, to 53% in 1985, to 37% in 1994, to a value between 32% and 35% (with mean 34%) in all later years. The community becomes larger, there are more nodes with high degree and thus it becomes harder to "know the friend of your friend". The average degree (number of researchers with whom you have at least one joint PODS paper) rises slowly from 1.21 (in 1982) via 2.23 (in 1991) and 2.83 (in 2001) to the current 3.38.

name	count	most central in
S. Abiteboul	14	89, 97, 99, 2001-2011
J. D. Ullman	7	87, 90–94, 98
C. H. Papadimitriou	6	82–85, 95, 2000
D. S. Parker Jr.	1	86
C. Beeri	1	88
A. Silberschatz	1	96

Table 2: Most central persons per year.

name	between. centrality
Serge Abiteboul	7%
Kenneth Ross	4%
S. Muthukrishnan	3%
Raghu Ramakrishnan	3%
Victor Vianu	3%
Christos H. Papadimitriou	3%
Jeffrey D. Ullman	3%
Abraham Silberschatz	3%
Dan Suciu	3%
Georg Gottlob	3%

Table 3: Ten most central PODS authors in 2011.

4.3 PODS communities

The set of PODS authors can be viewed as a Russian doll consisting of tighter and tighter communities. In a few steps, we reach a small core consisting mostly of prolific authors who were around since 1982. The obvious candidate for the first doll is the largest component (613 authors). For the next set of dolls we use a measure that has turned out to be very good in selecting subcommunities. It is a relaxation of the notion of a clique, called k-clique community [3]. Two k-cliques are considered adjacent if they share $k-1$ nodes. A community is then defined as the maximal union of k-cliques that can be reached from each other through a series of adjacent k-cliques.

The largest 3-clique community inside the largest component contains 125 nodes.[1] It has two largest 4-clique communities in it, both of size 15. These are disjoint, except for Ron Fagin. One of these components contains many prominent PODS authors. We have chosen this one as the next doll. This doll of 15 authors consists of a 5-clique and a 6-clique, both arising from a single paper[2] and four other authors.

On the PODS pages, we show a figure displaying

[1]The results in this section are calculated on the PODS 2010 graph which excludes the papers from PODS 2011.

[2]`http://www.informatik.uni-trier.de/~ley/db/conf/pods/pods96.html#AbiteboulKPV96` and `http://www.informatik.uni-trier.de/~ley/db/conf/pods/pods86.html#AfratiPPRSU86`.

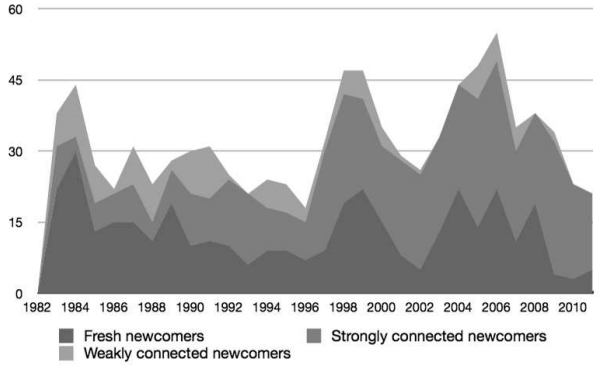


Figure 6: Number of newcomers per year.

the 125-node 3-clique community in blue with the two largest 15-node 4-clique communities it contains in pink and green. It also shows the next two largest 3-clique communities in green and orange. Looking at the names inside communities one cannot help but notice that cultural–geographic origin is a strong community–builder: Flanders and Greece–Israel provide strong binding force.

5 Newcomers

The number of newcomers per year is displayed in Figure 6. Newcomers are divided into three groups depending on their connection to the PODS graph of the year before. We have newcomers without any link to a PODS author ("fresh"), newcomers who are linked to a PODS author in the largest connected component ("strongly connected"), and those who are only connected to nodes outside the largest component ("weakly connected"). The average number of newcomers per year is 32 (over all years). Of these, on average, 40% are fresh, and 47% and 13% are strongly and weakly connected, respectively. Of these 32 newcomers, on average, 7.3 manage to get another PODS paper in the first three years after their first PODS paper. Just over half of these (3.9) even have at least two PODS papers in the first 5 years after their PODS-benjamin.

Which of the three groups of newcomers has the largest chance of reappearing in PODS? In other words, if you are a PODS newcomer, and intend to become a PODS regular, with whom should you write your first PODS paper? The results in Table 4 show that your chance or reemergence doubles when you, instead of starting fresh, join forces with a well-established PODS-member.

Figure 7: Newcomers for the years 2005-2011 according to their PODS score.

To get a feeling for the coming generation, Figure 7 displays the newcomers since 2005 according to their PODS score. On the PODS pages you can find a movie displaying the newcomers per year over the entire history of PODS.

6 Research Fields

PODS is influenced by and (hopefully) influences other fields of research. Such influence could be measured by examining the conference venues in which PODS-researchers publish. We give an overview how the relationship with other fields changes over time. To keep the analysis manageable, we restricted attention to the 100 venues in which PODS authors publish the most. Specifically, let p denote a conference paper by a PODS author. By $year(p)$ and $field(p)$, we denote the year of publication and the research field associated to p, respectively. Research field names were assigned uniformly for all papers in the same conference. For a research field f, a start year y_1 and an end year y_2, the influence of the PODS authors on (or from) field f during the period spanned by the interval $[y_1, y_2]$, denoted $influence(f, y_1, y_2)$, is defined as

$$|\{\text{paper } p \mid y_1 \leq year(p) \leq y_2 \wedge f = field(p)\}|\,.$$

Figure 8 displays the influence for three 10-year periods of time. While the method of computing influence can be criticized, the figures do confirm some

	All	Strongly Connected	Weakly Connected	Fresh
1 paper in 3 years	22%	30%	20%	15%
2 papers in 5 years	12%	16%	12%	7%

Table 4: Chance of reappearance of PODS newcomers, with at least one paper in the three years after first PODS publication and at least two papers after five years. Averages taken over the years 1983 to 2008 (3 years) and 1983 to 2006 (5 years).

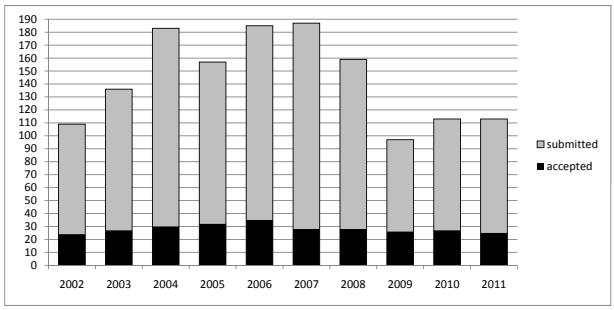


Figure 9: Number of PODS submissions and accepted papers per year for the period 2002–2011.

Figure 10: Locations of PODS in the period 2002–2011. The more submissions, the larger the font of the location.

global trends which on an intuitive level correspond to the reality. First of all, over time, the community became much more diverse as can be seen by the explosion of related research fields. Secondly, PODS researchers publish increasingly more papers outside the database theory community. Indeed, while the number of PODS-papers (or ICDT-papers for that matter) did not drastically change over the years, the database theory pie decreases from 15% to 5%, in favor of for instance, database systems which increased from 23 to 32%.

7 Conference statistics

Figure 9 displays the number of submissions as well as the number of accepted papers over the last ten years.

The average acceptance rate is 20%. Even though these figures are discussed at every PODS business meeting, there does not seem to be a satisfying explanation to why they are what they are. To provide an alternative view, Figure 10 relates submission numbers to conference locations.

8 Greek Gods of Python

During a divine intervention at the anniversary session two Greek gods suddenly materialized. The hymn they performed is depicted in Figure 11.

9 Outlook

There were several things we would liked to have investigated but where held back by constraints like lack of time and absence of data. Although the presented overview is mostly biased towards publications, we should not forget that the success of a community is also reflected by the number of people attending its main event even when they do not have a paper. It would be interesting to know how successfull our community is in this respect. Of course, as PODS merged with SIGMOD in 1996, the task no longer reduces to matching registrations to papers. We did not consider citations. Interestingly, DBLife [3] maintains a list of the top-300 most cited PODS-papers.

[3] http://dblife.cs.wisc.edu/

When your paper got rejected
by a review that corrected
your main result that turned out to be wrong.
Don't feel down, you're blessed!
It's Science that progressed.
So let's stand up and burst into a song.

And... Always look on the bright side of life...
Always look on the light side of life...

If your review pile just grows
to heights nobody knows
and the definition sections are unclear.
Don't let it make you mad.
Just think: "It's not so bad!
At least I didn't write the crap that's here."

And... Always look on the bright side of life...
Always look on the light side of life...

If you cannot comprehend
the talks that you attend
'cause they're just throwing formulas at you
and wireless is down.
Don't worry 'bout your frown.
No one knows that you don't have a clue

So... Always look on the bright side of life...
Always look on the light side of life...

If a student is so kind
to find a proof of just one line
for the central theorem of your PhD.
Don't feel bad or dumb
or that your life has gone:
You still were first, chronologically.

And... Always look on the bright side of life...
Always look on the light side of life...

If a talk you just gave
causes a big wave
of questions why your work is really new.
Wasn't it proved in 1960
by someone called Büchi?
Then at least you know that his result is true.

Always look on the bright side of life...
Always look on the right side of life...
Always look on the bright side of life...
Always look on the bright side of life...

Figure 11: Lyrics of "Always Look on the Bright Side of Life" (originally by Eric Idle), adapted for the 30th Anniversary of PODS.

Acknowledgements

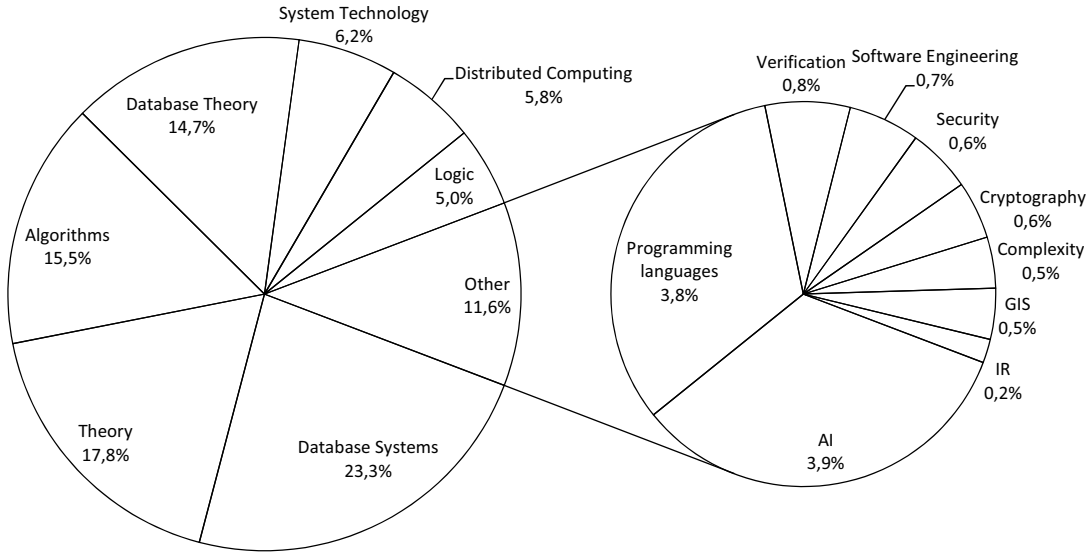
This analysis was only possible with the limited manpower we had because of the unsurpassed DBLP database created and maintained by Michael Ley.

References

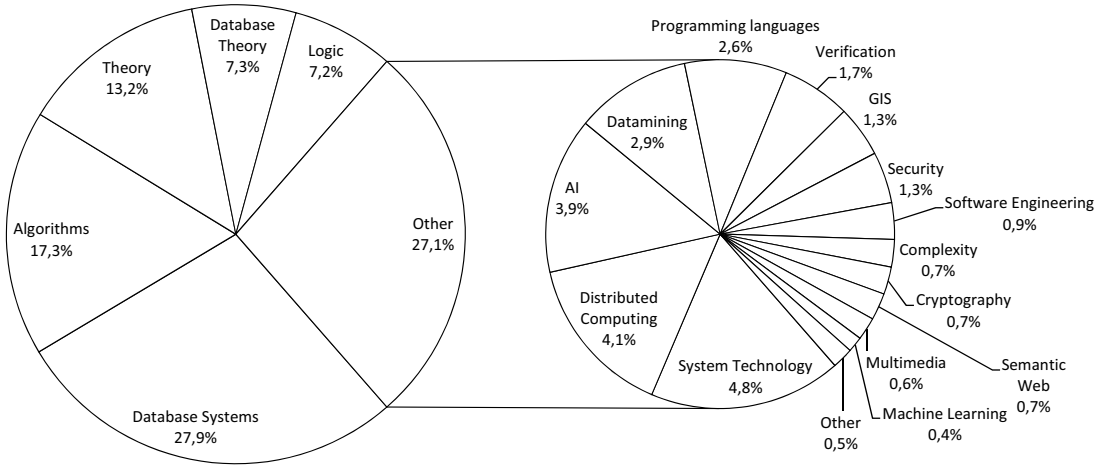
[1] The Pods Pages. Available at `http://www.sigmod.org/the-pods-pages`.

[2] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

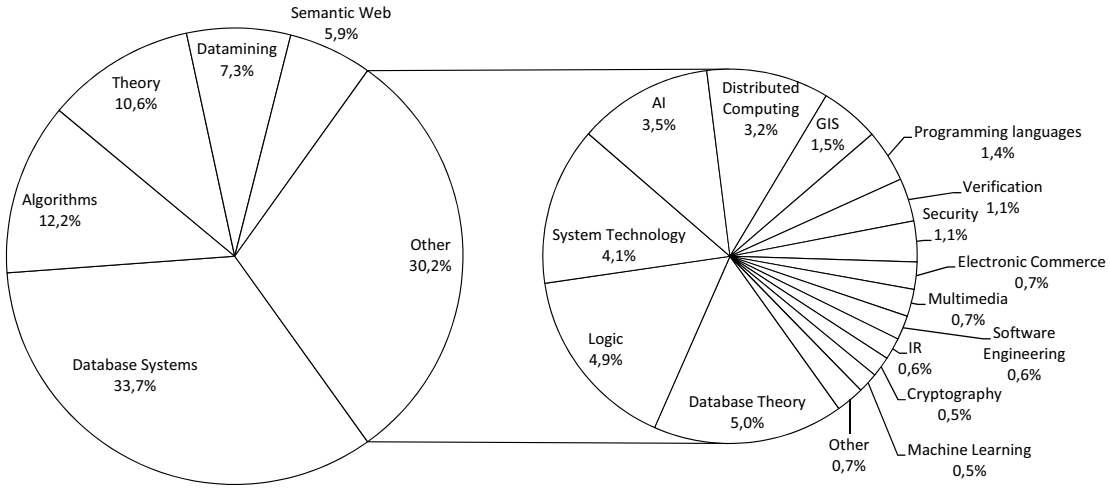
[3] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *NATURE*, 435:814–818, 2005.



(a) Period 1982-1991



(b) Period 1992-2001



(c) Period 2002-2011

Figure 8: Influence on research fields.

A Call to Arms: Revisiting Database Design

Antonio Badia
University of Louisville, USA
abadia@louisville.edu

Daniel Lemire
Université du Québec à Montréal, Canada
lemire@acm.org

1. INTRODUCTION

Good database design is crucial to obtain a sound, consistent database, and — in turn — good database design methodologies are the best way to achieve the right design. These methodologies are taught to most Computer Science undergraduates, as part of any Introduction to Database class [33]. They can be considered part of the "canon", and indeed, the overall approach to database design has been unchanged for years. Moreover, none of the major database research assessments identify database design as a strategic research direction [1, 2, 8].

Should we conclude that database design is a solved problem?

Our thesis is that *database design remains a critical unsolved problem*. Hence, it should be the subject of more research. Our starting point is the observation that traditional database design is not used in practice — and if it were used it would result in designs that are not well adapted to current environments [5]. In short, database design has failed to keep up with the times. In this paper, we put forth arguments to support our viewpoint, analyze the root causes of this situation and suggest some avenues of research. The point of view espoused here has been put forth more or less explicitly in other places (see [18] for a recent and notable example); but here we put together several strands that have received isolated attention, and focus them on an issue that we feel is particularly important — database design.

In the next section (§ 2), we sketch the traditional database design process: we argue that it manages to be, at the same time, *over-engineered* and *under-engineered*. The contradiction is only apparent: as any complex problem, this one is multifaceted. Traditional design does too little with respect to some areas and too much with respect to others. In § 3, we analyze the causes of the problems presented in § 2. We then briefly the current status of research on database design (§ 4). Finally, we present some ideas for a research renewal in § 5.

2. TRADITIONAL MODELING

Relational modeling is usually broken down into three steps:

- **Conceptual modeling**, which includes *requirement gathering and specification*, and results in a *conceptual model* of the database. At this stage, the designer focuses on issues of scope — *what belongs in the database?* — and organization — *how is the information to be structured?* Entity-relationship diagrams [15] and UML class models are the two best known conceptual models, but not the only ones; alternatives like Object Role Modeling have been proposed [29].
- **Logical modeling**, which takes as input the conceptual model produced in the previous step and yields a database schema. This step is well developed [60]. Normalization enforces *functional dependencies* by removing redundancy.
- **Physical modeling**, which takes as input the database schema produced in the previous step and produces storage structures to implement the schema in computer systems. It can be automated to a large extent [13, 24].

Each step focuses on only one aspect of the problem which helps tame the complexity. Also, each step produces an output that feeds into the next step, creating a linear structure that is easy to follow.

2.1 Problems with the traditional approach

The problem of database design is difficult, and it encompasses issues that may not be amenable to formalization [52]. Hence, any method is likely to have some limitations and drawbacks. However, this is not a reason to ignore the serious problems that the traditional approach is running into. Here

we summarize what we see, from our experience and perspective, as the most troublesome issues.

2.1.1 Failures of use and guidance

We claim that the traditional approach is not followed in practice. Indeed, Fitzgerald et al. [23] found that only about 11% of the consulted organizations claimed using an unmodified commercial information system methodology. Furthermore, Brodie and Liu [11] report that while 90% of all information systems inside a Fortune 100 company are relational, they could not find a single instance of an entity-relationship modeling in over ten such large corporations. The lack of modeling is not due to the lack of complexity: they report that a typical Fortune 100 company has about 10 thousand different information systems, that a typical relational database is made of over 100 tables, each containing between 50 to 200 attributes. Formalized conceptual models, as well as the theory developed around normalization, are not used. Physical modeling is frequently delayed until performance problems arise. In a very real way, we have entered a post-methodological era as far as the design of information systems is concerned [5]. The emergence of the Web has coincided with the death of the dominant methods based on the analytic thought and lead to the emergence of sense-making as a primary paradigm.

If one agrees that the traditional method is not used, the obvious question is: Why? Why do practitioners dismiss a method that has a solid theoretical basis and is the distillation of years of thought? It would be easy (and tempting!) to blame the designers or their training. But the tools themselves share a good part of the blame. They fail to give what designers need most, *guidance as to how to apply them*: for conceptual models, not enough guidance is given as how to create one, how to assess its quality, and — importantly — how to handle all information that does not fit into the conceptual model but may be relevant later for data quality of other purposes.

A critical failure in the traditional approach is that there is little guidance on how to discover important information (e.g., functional dependencies) in the real world. It would not be a concern if the rest of the design assumed that we lacked information. Yet unless we have *all* functional dependencies, there is no guarantee of normal form in the logical design. Thus, the logical design phase is *brittle*.

Ironically, the step where most research has focused on giving guidance is the last one, physical design [13], perhaps because it is easier to simulate realistically the problems and their solutions in a laboratory. However, this third step relies on the previous ones; while it can sometimes result in modifications of the database schema — as when denormalization is recommended, most approaches still assume that a schema has been well designed. By analogy, we could say that we know how to build the walls, as long as the foundation of the house is, somehow, done properly.

2.1.2 Failures of imagination

Even if one were to follow the steps of the traditional design method, and have a perfectly normalized, by-the-book database, what does one obtain?

We consider database design a matter of semantics: we are trying to capture the semantics of a domain, to represent information about that domain faithfully, and to (only) allow operations with the data that are meaningful. But traditional database design focuses on structure. In exchange for all the effort, we have insufficient semantics. This is the sense in which databases are *under-engineered*.

Consider, for instance, the problem of *information integration* [20, 54, 63]. Relational databases fail to provide enough information to determine automatically whether two databases contain information about distinct, overlapping, or similar domains. And yet, integration of information is increasingly critical: 40% of the cost associated with information systems is due to data integration problems [11]. To exemplify this trend toward greater integration and collaboration even in the most conservative settings, consider that the 9/11 Commission report urged the intelligence community to move from its need-to-know standard to a need-to-share approach [35]. Experts believe that the 9/11 tragedy could have been avoided with better data integration. The traditional way to design databases does not capture enough information to enable information integration — in fact, it falls short of capturing precisely the kind of information that would be more valuable for integration. Hence, traditional design not only fails to alleviate the problem, *it is helping to perpetuate it*. Most data integration approaches start by trying to determine the similarity between attributes.[1] Since most design approaches treat attributes as barely more than labels, one has usually only a string to work with: information *about* at-

[1]Several approaches rely on statistical properties of data, and choose not to try to interpret it [36]. It is unclear whether this is done in search of generality or due to need; but we believe that, while this approach provides important information, statistical properties cannot *establish* semantic similarity *by themselves*—but see Halevy et al. [28] for a different viewpoint.

tributes (metadata) is usually absent [53]. As long as design focuses on how to structure attributes in tables and not in what attributes mean, the problem will be with us. In the end, we ask practitioners to follow a model that is demanding and yields, in return, some very limited results.

The lack of appropriate metadata is even more acute in new applications, ranging from financial to legal systems. A prominent example is *e-science*: scientists need not only to store larger and larger amounts of data. They also need to be able to assess provenance [57], access rights, workflows, etc. in order to comply with ever increasing regulations, to be able to share the data, and to achieve the goal of *reproducible research* [58]. On this, traditional design offers no guidance.

To make matters worse, the focus on structure creates rigidity. Kiely and Fitzgerald [37] found that traditional information systems development methods were sometimes perceived to be of limited use within modern projects because they are too cumbersome and inflexible. This is the sense in which databases may be considered *over-engineered*. Consider the NoSQL movement [40]. A large force behind it are programmers for which database design makes no sense. Tired of the rigid structure of relational databases, other systems (Raven DB,[2] Amazon's SimpleDB,[3] Apache's CouchDB,[4] MongoDB[5]) are emerging. What good is it to design if it fails to make the developers more productive? Unfortunately, the mismatch between objects and program structures on one hand, and database structures on the other, is still largely unresolved. Motivated by this problem, Microsoft has proposed the Language-Integrated Query (LINQ) framework [44]. Other initiatives to bridge the gap have been developed over the years — witness to the fact that the problem is still with us.

3. WHY DOES IT FAIL?

The traditional design method was developed in the early seventies, when mainframes dominated information technology. It is in this era that the relational [17] and entity-relationship models [15] were invented. Accordingly, there are several assumptions behind the traditional design which reflect its age:

• *Users are faceless objects for whom (or on whose behalf) the systems are designed [31].*[6] In the seventies, the management of data was left in the hands of few experts who served the needs of technologically unsophisticated employees. Nowadays, the boundaries between users, whether they are employees or clients, and developers are blurred [46]. This is best illustrated with how *hashtags* emerged on the microblogging platform Twitter. Hashtags are a metadata convention among Twitter users [39], in the spirit of folksonomies [55]. Yet Twitter itself had no support for metadata. We can trace back the current convention to a single user who informally proposed it in a 140-character post in August 2007. Later, Twitter engineers recognized the convention and added software support for it. For example, Twitter detects "trending topics" using popular hashtags. A few other conventions, like the "retweet" were first initiated by the users. Sundara Nagarajan has recently expressed the same idea [48]: "Empowered end users cause application systems to evolve at tremendous speeds and continuously create new requirements for interoperation. For instance, a social networking site user can add content and pointers from a website, by simply dragging and dropping. The evolution of mashups that combine data and functionality from multiple sources is another example of this new design paradigm. This is leading to the evolution of the user experience, along with computation and data management." When systems are designed without the users, a lack of user engagement may result: 93% of all accounts in Business Intelligence systems are never used [45].

• *The information system is strongly consistent.* It has been estimated that Google alone has more than 1 million servers. Using cloud computing, anyone can use a distributed network of servers at a modest cost. With multiply located servers and deeply integrated web services, the CAP theorem [25] implies in practice that we have to choose between strong consistency and strong availability: we cannot have both. As a possible illustration of this constraint, the recent failure of an Oracle database at JPMorgan Chase, which froze $132 million in assets and lost thousands of loan applications, was blamed on an database [47] which required strong consistency for all data.

• *Semantics is absolute.* The original design assumed a centralized architecture. This architectural assumption had a reflection on the conceptual level, where one main viewpoint was assumed. While semantic relativism is pointed out [21, 51, 56], a choice must be made for a single model: there is no mech-

[2] `http://ravendb.net`
[3] `http://aws.amazon.com/simpledb/`
[4] `http://couchdb.apache.org/`
[5] `http://www.mongodb.org/`
[6] In the quote sometimes attributed to Frederick the Great, it's "everything for the people, but without the people".

anisms to derive other. Yet when different systems must interact routinely, we cannot expect that they all share the same viewpoint.

• *The models are static.* In the traditional setting, there is little need for evolution. Yet databases, even in large conventional corporations, are fast evolving: 30% of all information systems are modified significantly every year [11] in Fortune 100 companies. Chen, the father of the entity-relationship model, recently recognized the difficulty by pointing out the inability of existing modeling techniques to cope with fast-varying world states [16].

4. WHAT NOW?

Despite these difficulties, *research on database design has failed to make major progress in the last ten years or so.* This is not to say that no research is done. For instance, the series of conferences on entity-relationship modeling [50], while not totally focused on design issues, devote most of the program to them. Theoretical work is still ongoing in logical modeling [38]. Physical design research is still strong, sometimes driven by database vendors [3]. And there is still a community of dedicated researchers including notable researchers like Thalheim [61] and Olivé [49] among others.

But overall the topic is not widely pursued. For instance, if one checks the last 10 years of the "major" database conferences (SIGMOD, VLDB, ICDE), the number of papers in database design is low: leaving aside *physical design* (that is, the fine tuning of storage structures for better performance), we found less than ten talks *mainly* about database design [4, 9, 26, 34, 41, 42, 58, 62] and most of them examine design issues within the confines of a restricted context (sensor databases [41, 42], XML schema evolution [9], user interaction [62], scientific databases [58], data warehousing [34]).[7] There are certainly papers which, without having design as their main goal, bring considerable contributions to the table. For instance, the research on *dataspaces* by Halevy et al. [27] has brought forth the possibility of databases where the schema is implicit or at least not separated from the data and can evolve with it. It opens up some possibilities, but no paper on this project is about design *per se*. Likewise, research on semistructured data (e.g., XML) has exposed the database community to the thought that design must be more flexible [6]. However, little of this seems to have percolated to more traditional (relational, object relational) data models, and the design methodology for them. Finally, the total number of contributions remain low, even including this work, for such a crucial topic.

The fact is, *traditional database design is not a mainstream research topic.* We believe that this is due to two main facts: first, for most researchers, work on database conceptual models is seen as too difficult, because the subject is "soft", not clearly formalized, and does not yield itself well to the typical paper that one expects to see published in most conferences and technical journals. Second, work on relational design is considered useless as the topic is commonly taken as basically a mature and closed one. Certainly, there is always some more work that can be done (for instance, extending the idea of key and functional dependence to other models, like XML, has received some recent attention [14], as well as extending the concept itself to 'soft' functional dependencies [32]), but the subject is often considered "a solved problem".

On teaching, we note that while some textbooks are quite good about pointing out to students the limits and difficulties of the process, others simply gloss over the issues and give the impression that this a "case closed" situation — which may contribute to the lack of research in the area.

5. WHAT NEXT?

Traditional database design fails to provide the tools needed to design databases in today's environment, but researchers have not updated or expanded the methodologies enough to keep up with the times. Should we continue teaching methodologies which disappoint practitioners?

A first step towards renewed emphasis on database design research is to come up with a fresh and timely

[7] We do not claim any statistical validity for this observation. For one, the sample used is limited — more conferences, and certainly some journals, should be included. For another, there is a subjective aspect to this analysis. For the sake of transparency, we explain our method: first, the web page of each conference, reached through the web site of the organization behind the conference was used: this gives access to session titles, paper titles and sometimes abstracts. A search was made for keyword "design", another for keyword "normal" (to obtain 'normalization' and so on), and another for "semantic". We checked the title and abstract for each match (the last keyword generated quite a few matches). As stated, papers on physical design (database tuning and index design) were excluded. The list of papers obtained is given for interested readers to judge by themselves in the references above. Others may reasonably disagree exactly as to what to include, i.e., how to define 'mainly about database design' (but note that we include an invited talk and two tutorials!). However, unless one uses a generous notion of 'database design', we believe other people's results will be in the same order of magnitude as ours.

approach. Different researchers will likely have different viewpoints as to what are the most crucial or interesting problems. We submit the following research plan to open up a discussion.

Design for a distributed world.

- We must update database design methodologies for new environments that did not exist in the 1970s. Though there were many failed attempts to replace the ACID-compliant relational database systems with *better* alternatives, the landscape has finally begun changing with the adoption of cloud computing. For example, the data consistency requirements (and other issues affecting distribution) should be made explicit during the design phase, so that they can be exploited when deciding an architecture. In fact, many NoSQL designs assume that most operations can be kept local in order to ensure scalability, which means that one needs to know which data is likely to be involved in a transaction (logically related) in order to distribute the data ([59] makes the same point, implicitly). Along the same lines, deciding what can be made eventually consistent (versus what needs to be kept consistent at all times), and what to do in the face of inconsistency, should be based on the semantics of data. Hence, such issues should be part of the design phase.

- It is fashionable to talk about Big Data: one the main driver being this trend is our ability to quickly integrate diverse data sets to create new services. Correspondingly, *easier data integration should become one of the primary goals of good database design.* Another issue that Big Data brings is the distributed nature of the model. Do we need a 'distributed design' approach? For instance, should design produce more or less independent modules or 'chunks' of connected data, which can in turn be connected to each other in one or more ways? How would such a distributed design relate to Berners-Lee's linked data [7]? Or perhaps we should propose methodologies which, instead of starting from a clean slate, begin with the existing schemas (both within the organization, and public ones) and build on top of them. Should we shift the focus towards *extensions of what there is*?

Rethink functional dependencies.

- If Helland is right and normalization is for sissies ([30]), then one should question the focus on functional dependencies in database design. If this idea seems far-fetched, recall that data warehousing practitioners proposed a different design methodology (the star schema) that does not use the idea of functional dependency at all (rationalization of star schema in normal form came after the fact). The question then is whether there are other concept or concepts that can replace 'functional dependence' and be a good basis for design.

- We know that enforcing functional dependencies in the schema is insufficient to ensure that the data is semantically consistent. There are many rules, some expressible as constraints, assertions, or triggers, that could be enforced to ensure meaningful data. But current design mostly ignores this information. Shouldn't we attempt to capture this information, which is most likely to have an impact on the quality of our data, during the design? (In which case we need to define a way to measure the impact of different types of rules in data quality and consistency.) How can these various rules be used *together* in design? Note that to answer this question one has to answer other, more basic, questions: how do these different rules interact?

- Much of the database-design courses focus on functional dependence and normal form. It is often implied that the physical design ought to be a straight-forward application of the logical design. This is because, once, the equation *one relation = one table = one file* held for virtually all relational systems. Yet it not longer applies. For example, many distributed or column-oriented database systems replicate data for speed or reliability. Is it time to *completely* separate logical design from physical design, i.e. consider the relation as a purely conceptual entity?

Design for imperfect knowledge.

- We must cope with incomplete information (about the domain, the users, etc.) since in real systems, the scope or boundary of a database, or its future usage, is often uncertain [43]. Thus, design should proceed with as few assumptions as possible. Until now, a certain *closed world assumption* mentality trickles all the way from the conceptual model to the database. Clearly,

we live in an *open* world. Should be consider schemas as *descriptive* instead of *prescriptive*, which is what they are now? If so, what to do with data that does not follow the schema? Should any such data be allowed? Given the difficulty of determining in advance the type of data that the system may have to deal with, should the design include, for instance, a description of data that should *not* be allowed, and leave the database open to all other data? To some extend, XML schema languages (e.g., XML Schema and Relax NG) seem to adopt such a permissive attitude, with the added requirement that the data be structured in a hierarchical manner. Unfortunately, our experience is that the process is burdensome and is not widely used [10]. Are there lightweight alternatives?

- In turn, adopting an open world point of view will make it easier to support collaborative, evolutionary design as an integrated part of the workflow [18]. The issue here is, how do we design databases with *open*-world model while insuring the necessary consistency? If we are going to give permission to users to modify a schema, how much freedom should users have? For instance, one could study *whether design can be crowdsourced* (and if so, how and under what constraints). In general, one need to decide what kind of changes can be support, whether they come from the users or from a designer. A deeper study of *database evolution* could be of help here: could a system be designed that adapts its storage to changing schemas and requirements? Physical design is currently focused on *query workload*, that is, it adapts itself to the (changing) requirements posed by the database queries. Could some of these ideas be used to make the system reactive to changes in the schema? We find interesting that functional dependencies can be (roughly) classified as *natural* (one that reflects an invariant in the world: a person has only one height) or *artificial* (one that reflects a convention: each employee has to attend X meetings a month). The former are quite stable, but the latter are subject to change (note that all so-called *business rules* are artificial!). Should a system be able to cope with changes in artificial dependencies (old ones cease to hold, new ones are added)?

- New data stores in the NoSQL movement use non-relational data models: key/value, documents, extensible records [12]. Probably the first research task for such data models is a clarification of their exact structure and properties, since the terms are used somewhat loosely. But an immediate second is to decide whether they require a different approach to design (after all, even NoSQL data stores require design) or, to the contrary, whether design decisions can be kept independent of the data model. The question is not as trivial as it may seem: some of these new models allow an *open schema*, that is, one where the user can add attributes at will, while others still require, like relational databases, a *closed schema*, that is, one where all possible attributes are declared beforehand — yet others, like extensible records, combine both parts.

- Though there has been much work done on probabilistic databases [19] and soft functional dependencies [32], such subjects remain almost entirely distinct from database design. Yet semantics are not always absolute: some relationships are merely almost always true. Thus, it is likely that there are many more soft dependencies or conditional dependencies than 'standard' functional dependencies. (A conditional dependency is one that holds only under certain circumstances. For example, at some places, a married couple is always made of a man and a woman, but not at others.) Current design practices tend to ignore all functional dependencies but the standard ones, which are but an extreme case [22]. Should we make room in database-design methodologies for probabilistic metadata and several types of dependencies? If so, how would different types of dependencies be used? How would they behave when put together?

No doubt, different researchers will have different viewpoints on these issues. Some may object to some of the challenges included here; others may wish to direct attention to other problems not included here. We stress again that this plan is meant to start the discussion; let the debate begin.

References

[1] S. Abiteboul et al. The Lowell database research self-assessment. *Communications of the ACM*, 48:111–118, May 2005.

[2] R. Agrawal et al. The Claremont report on database research. *SIGMOD Record*, 37:9–19, September 2008.

[3] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for Microsoft SQL Server 2005: demo. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 930–932, New York, NY, USA, 2005. ACM.

[4] P. Andritsos, R. J. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 731–742, New York, NY, USA, 2004. ACM.

[5] D. E. Avison and G. Fitzgerald. Where now for development methodologies? *Communications of the ACM*, 46:78–82, January 2003.

[6] D. Barbosa, J. Freire, and A. O. Mendelzon. Designing information-preserving mapping schemes for XML. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 109–120. VLDB Endowment, 2005.

[7] T. Berners-Lee. Linked data. *International Journal on Semantic Web and Information Systems*, 4(2), 2006.

[8] P. Bernstein et al. The Asilomar report on database research. *SIGMOD Record*, 27:74–80, December 1998.

[9] K. Beyer, F. Özcan, S. Saiprasad, and B. Van der Linden. DB2/XML: designing for evolution. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 948–952, New York, NY, USA, 2005. ACM.

[10] T. Bray. Don't Invent XML Languages. `http://bit.ly/364VEy` [last checked on 10/07/2011], 2006.

[11] M. L. Brodie and J. T. Liu. The power and limits of relational technology in the age of information ecosystems. Keynote at On The Move Federated Conferences, 2010.

[12] R. Cattell. Scalable SQL and NoSQL data stores. *Sigmod Record*, 39(2):12–27, 2010.

[13] S. Chauduri and V. Nrasayya. Self-tuning database systems: A decade of progress. In *Proceedings of VLDB*. VLDB Endowment, September 2007.

[14] H. Chen, H. Liao, and Z. Gao. Functional dependencies for XML. In *Proceedings of the 2010 international conference on Web-age information management*, WAIM'10, pages 110–115, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[16] P. Chen. Suggested research directions for a new frontier: Active conceptual modeling. In D. Embley, A. Olivé, and S. Ram, editors, *Conceptual Modeling - ER 2006*, volume 4215 of *Lecture Notes in Computer Science*, pages 1–4. Springer Berlin / Heidelberg, 2006.

[17] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[18] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. MAD skills: new analysis practices for big data. *Proc. VLDB Endow.*, 2:1481–1492, August 2009.

[19] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52:86–94, July 2009.

[20] A. Doan, P. Domingos, and A. Y. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3), 2003.

[21] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 6th edition, 2010.

[22] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23:683–698, 2011.

[23] G. Fitzgerald, A. Philippides, and S. Probert. Information systems development, maintenance and enhancement: findings from a UK study. *International Journal of Information Management*, 19(4):319–328, 1999.

[24] K. E. Gebaly and A. Aboulnaga. Robustness in automatic physical database design. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 145–156, New York, NY, USA, 2008. ACM.

[25] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):59, 2002.

[26] L. Golab, H. J. Karloff, F. Korn, and D. Srivastava. Data auditor: Exploring data quality and semantics using pattern tableaux. *PVLDB*, 3(2):1641–1644, 2010.

[27] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 1–9, New York, NY, USA, 2006. ACM.

[28] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[29] T. Halpin. *Conceptual Schema and Relational Database Design*. Prentice Hall, 1989.

[30] P. Helland. Normalization is for sissies. In *Conference on Innovative Database Systems Research (CIDR)*, 2009.

[31] J. Iivari, H. Isomäki, and S. Pekkola. The user, the great unknown of systems development: reasons, forms, challenges, experiences and intellectual contributions of user involvement. *Information Systems Journal*, 20(2):109–117, 2010.

[32] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 647–658, New York, NY, USA, 2004. ACM.

[33] J. Impagliazzo. Computing curricula 2005. *ACM SIGCSE Bulletin*, 38(3):311–311, 2006.

[34] M. Jarke, C. Quix, D. Calvanese, M. Lenzerini, E. Franconi, S. Ligoudistianos, P. Vassiliadis, and Y. Vassiliou. Concept based design of data warehouses: the DWQ demonstrators. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 591–, New York, NY, USA, 2000. ACM.

[35] C. Jones. Intelligence reform: The logic of information sharing. *Intelligence & National Security*, 22(3):384–401, 2007.

[36] J. Kang and J. Naughton. Schema matching using interattribute dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 20(10), October 2008.

[37] G. Kiely and B. Fitzgerald. An investigation of the use of methods within information systems development projects. *The Electronic Journal of Information Systems in Developing Countries*, 22, 2005.

[38] S. Kolahi and L. Libkin. An information-theoretic analysis of worst-case redundancy in database design. *ACM Transactions on Database Systems*, 35:5:1–5:32, February 2008.

[39] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[40] N. Leavitt. Will NoSQL databases live up to their promise? *Computer*, 43(2):12–14, 2010.

[41] Q. Luo and H. Wu. System design issues in sensor databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1182–1185, New York, NY, USA, 2007. ACM.

[42] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.

[43] M. Magnani and D. Montesi. A survey on uncertainty management in data integration. *J. Data and Information Quality*, 2:5:1–5:33, July 2010.

[44] E. Meijer, B. Beckman, and G. Bierman. LINQ: reconciling object, relations and XML in the .NET framework. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 706–706, New York, NY, USA, 2006. ACM.

[45] R. Meredith and P. O'Donnell. A functional model of social media and its application to business intelligence. In *Proceeding of the 2010 conference on Bridging the Socio-technical Gap in Decision Support Systems: Challenges for the Next Decade*, pages 129–140, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[46] F. Millerand and K. Baker. Who are the users? Who are the developers? Webs of users and developers in the development process of a technical standard. *Information Systems Journal*, 20(2):137–161, 2010.

[47] C. Monash. Details of the JPMorgan Chase Oracle database outage. `http://bit.ly/ckWLfq` [last checked on 10/07/2011], 2010.

[48] S. Nagarajan. Guest editor introduction. *Data Storage Evolution, Special Issue of Computing Now*, March 2011. IEEE Press.

[49] A. Olivé. *Conceptual Modeling of Information Systems*. Springer, 2007.

[50] J. Parsons, M. Saeki, P. Shoval, C. C. Woo, and Y. Wand, editors. *Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling*, Lecture Notes in Computer Science 6412, Vancouver, BC, Canada, November 2010. Springer.

[51] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2002.

[52] J. F. Roddick, A. Ceglar, D. De Vries, and S. La-Ongsri. Active conceptual modeling of learning. In P. Chen and L. Y. Wong, editors, *Postponing schema definition: low instance-to-entity ratio (LItER) modelling*, pages 206–216, Berlin, Heidelberg, 2007. Springer-Verlag.

[53] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to falilitate interoperability among heterogeneous information systems. *ACM Trans. Database Syst.*, 19(2):254–290, 1994.

[54] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. Tech report dit-04-087, University of Trento, 2005. Also: Journal of Data Semantics, LNCS 3730, pp. 146-171, 2005.

[55] S. Siersdorfer and S. Sizov. Social recommender systems for web 2.0 folksonomies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, HT '09, pages 261–270, New York, NY, USA, 2009. ACM.

[56] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 6th edition, 2010.

[57] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34:31–36, September 2005.

[58] E. Stolte, C. von Praun, G. Alonso, and T. Gross. Scientific data repositories: designing for a moving target. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 349–360, New York, NY, USA, 2003. ACM.

[59] M. Stonebraker and R. Cattell. 10 rules for scalable performance in 'simple operations' datastores. *Communications of the ACM*, 54(6):72–80, 2011.

[60] T. J. Teorey, S. S. Lightstone, T. Nadeau, and H. V. Jagadish. *Database Modeling and Design, Fifth Edition: Logical Design*. Morgan Kaufmann, 5th edition, 2011.

[61] B. Thalheim. *Fundamentals of Entity-Relationship Modeling*. Springer-Verlag, 2000.

[62] D. Tunkelang. Design for interaction. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 969–970, New York, NY, USA, 2009. ACM.

[63] O. Udrea, L. Getoor, and R. J. Miller. Leveraging data and structure in ontology integration. In *Proceedings of SIGMOD*, pages 449–460, 2007.

SIGMOD 2012 CALL FOR RESEARCH PAPERS

ACM SIGMOD International Conference on Management of Data

Scottsdale, Arizona, USA

May 20-25, 2012

http://www.sigmod.org/2012/

General Chairs:
K. Selcuk Candan (Arizona State University)
Yi Chen (Arizona State University)
General Vice-Chair:
Richard Snodgrass (University of Arizona)
Program Chair:
Luis Gravano (Columbia University)
Program Committee Group Leaders:
Anastasia Ailamaki (EPFL)
Philip Bernstein (Microsoft Research)
Elisa Bertino (Purdue University)
Umeshwar Dayal (HP Labs)
Juliana Freire (NYU-Poly)
Minos Garofalakis (Technical University of Crete)
Donald Kossmann (ETHZ)
Tova Milo (Tel Aviv University)
Divesh Srivastava (AT&T Labs-Research)
Gerhard Weikum (Max-Planck Institute for Informatics)
Proceedings Chair:
Ariel Fuxman (Microsoft Research)
Tutorial Chair:
Alon Halevy (Google Research)
Keynote and Panel Chair:
Surajit Chaudhuri (Microsoft Research)
Industrial Program Chair:
AnHai Doan (University of Wisconsin-Madison and @WalmartLabs)
Demonstration Chair:
Magdalena Balazinska (University of Washington)
Workshop Chair:
Christian S. Jensen (Aarhus University)
Undergraduate Research Program Chair:
Christopher Ré (University of Wisconsin-Madison)
Finance Chairs:
Egemen Tanin (University of Melbourne)
Junichi Tatemura (NEC Labs)
Publicity/Social Media Chairs:
Lei Chen (HKUST)
Maria Luisa Sapino (University of Torino)
Sponsorship Chairs:
Divyakant Agrawal (University of California at Santa Barbara)
Vasilis Vassalos (Athens U. of Economics and Business)
Exhibit Chairs:
Wen-Syan Li (SAP)
Berthold Reinwald (IBM Almaden Research Center)
Local Arrangements Chair:
Suzanne Dietrich (Arizona State University)
Registration Chairs:
Ziyang Liu (NEC Labs)
Bongki Moon (University of Arizona)
Demonstration and Workshop Local Arrangements Chair:
Hasan Davulcu (Arizona State University)
Web/Information Chairs:
Huiping Cao (New Mexico State University)
Yan Qi (Turn Inc.)

The annual ACM SIGMOD conference is a leading international forum for database researchers, practitioners, developers, and users to explore cutting-edge ideas and results, and to exchange techniques, tools, and experiences. We invite the submission of original research contributions relating to all aspects of data management defined broadly, and particularly encourage submissions on topics of emerging interest in the research and development communities.

TOPICS OF INTEREST

Topics of interest include but are not limited to the following:

- Benchmarking and performance evaluation
- Data analytics
- Data cleaning, integration, and provenance
- Data mining and knowledge discovery
- Data models, semantics, and query languages
- Data privacy and security
- Data streams and sensor networks
- Data visualization
- Database monitoring and tuning
- Databases for emerging hardware
- Distributed and parallel databases
- Indexing and physical database design
- Information extraction
- Information retrieval and text mining
- Mobile databases
- Modeling approximation and uncertainty in databases
- Query processing and optimization
- Scientific databases
- Semi-structured data
- Service-oriented computing and cloud data management
- Social networks and graph databases
- Storage systems
- Transaction management

SUBMISSION GUIDELINES

All aspects of the submission and notification process will be handled electronically. Submissions must adhere to the paper formatting instructions. Research papers will be judged for quality and relevance through double-blind reviewing, where the identities of the authors are withheld from the reviewers. Thus, author names and affiliations must not appear in the papers, and bibliographic references must be adjusted to preserve author anonymity. Submissions should be uploaded at https://cmt.research.microsoft.com/SIGMOD2012/.

IMPORTANT DATES

October 25, 2011, 5 p.m. PDT: Research paper abstracts due
November 1, 2011, 5 p.m. PDT: Research papers due
January 19—January 26, 2012, 5 p.m. PST: Author feedback accepted
February 14, 2012: Notification of acceptance