SIGMOD Officers, Committees, and Awardees

Chair
Juliana Freire
Computer Science & Engineering
New York University
Brooklyn, New York
USA
+1 646 997 4128
juliana.freire <at> nyu.edu

Vice-Chair
Ihab Francis Ilyas
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario
CANADA
+1 519 888 4567 ext. 33145
ilyas <at> uwaterloo.ca

Secretary/Treasurer
Fatma Ozcan
IBM Research
Almaden Research Center
San Jose, California
USA
+1 408 927 2737
fozcan <at> us.ibm.com

SIGMOD Executive Committee:
Juliana Freire (Chair), Ihab Francis Ilyas (Vice-Chair), Fatma Ozcan (Treasurer), K. Selçuk Candan, Yanlei Diao, Curtis Dyreson, Christian S. Jensen, Donald Kossmann, and Dan Suciu.

Advisory Board:
Yannis Ioannidis (Chair), Phil Bernstein, Surajit Chaudhuri, Rakesh Agrawal, Joe Hellerstein, Mike Franklin, Laura Haas, Renee Miller, John Wilkes, Chris Olsten, AnHai Doan, Tamer Özsu, Gerhard Weikum, Stefano Ceri, Beng Chin Ooi, Timos Sellis, Sunita Sarawagi, Stratos Idreos, Tim Kraska

SIGMOD Information Director:
Curtis Dyreson, Utah State University

Associate Information Directors:
Huiping Cao, Manfred Jeusfeld, Asterios Katsifodimos, Georgia Koutrika, Wim Martens

SIGMOD Record Editor-in-Chief:
Yanlei Diao, University of Massachusetts Amherst

SIGMOD Record Associate Editors:
Vanessa Braganholo, Marco Brambilla, Chee Yong Chan, Rada Chirkova, Zachary Ives, Anastasios Kementsietsidis, Jeffrey Naughton, Frank Neven, Olga Papaemmanouil, Aditya Parameswaran, Alkis Simitsis, Wang-Chiew Tan, Pinar Tözün, Marianne Winslett, and Jun Yang

SIGMOD Conference Coordinator:
K. Selçuk Candan, Arizona State University

PODS Executive Committee:
Dan Suciu (Chair), Tova Milo, Diego Calvanse, Wang-Chiew Tan, Rick Hull, Floris Geerts

Sister Society Liaisons:
Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE).

Awards Committee:
Martin Kersten (Chair), Surajit Chadhuri, David DeWitt, Sunita Sarawagi, Mike Carey

Jim Gray Doctoral Dissertation Award Committee:
Ioana Manolescu (co-Chair), Lucian Popa (co-Chair), Peter Bailis, Michael Cafarella, Feifei Li, Qiong Luo, Felix Naumann, Pinar Tozun

SIGMOD Systems Award Committee:
Mike Stonebraker (Chair), Make Cafarella, Mike Carey, Yanlei Diao, Paul Larson

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. Recipients of the award are the following:

Michael Stonebraker (1992)	Jim Gray (1993)	Philip Bernstein (1994)
David DeWitt (1995)	C. Mohan (1996)	David Maier (1997)
Serge Abiteboul (1998)	Hector Garcia-Molina (1999)	Rakesh Agrawal (2000)
Rudolf Bayer (2001)	Patricia Selinger (2002)	Don Chamberlin (2003)
Ronald Fagin (2004)	Michael Carey (2005)	Jeffrey D. Ullman (2006)
Jennifer Widom (2007)	Moshe Y. Vardi (2008)	Masaru Kitsuregawa (2009)
Umeshwar Dayal (2010)	Surajit Chaudhuri (2011)	Bruce Lindsay (2012)
Stefano Ceri (2013)	Martin Kersten (2014)	Laura Haas (2015)
Gerhard Weikum (2016)	Goetz Graefe (2017)	Raghu Ramakrishnan (2018)

SIGMOD Systems Award

For technical contributions that have had significant impact on the theory or practice of large-scale data management systems.

Michael Stonebraker and Lawrence Rowe (2015) Martin Kersten (2016) Richard Hipp (2017)
Jeff Hammerbacher, Ashish Thusoo, Joydeep Sen Sarma; Christopher Olston, Benjamin Reed, Utkarsh Srivastava (2018)

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)	Gio Wiederhold (1995)	Yahiko Kambayashi (1995)
Jeffrey Ullman (1996)	Avi Silberschatz (1997)	Won Kim (1998)
Raghu Ramakrishnan (1999)	Michael Carey (2000)	Laura Haas (2000)
Daniel Rosenkrantz (2001)	Richard Snodgrass (2002)	Michael Ley (2003)
Surajit Chaudhuri (2004)	Hongjun Lu (2005)	Tamer Özsu (2006)
Hans-Jörg Schek (2007)	Klaus R. Dittrich (2008)	Beng Chin Ooi (2009)
David Lomet (2010)	Gerhard Weikum (2011)	Marianne Winslett (2012)
H.V. Jagadish (2013)	Kyu-Young Whang (2014)	Curtis Dyreson (2015)
Samuel Madden (2016)	Yannis E. Ioannidis (2017)	Z. Meral Özsoyoğlu (2018)

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.* Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau. *Honorable Mentions*: Marcelo Arenas and Yanlei Diao.
- **2007** *Winner*: Boon Thau Loo. *Honorable Mentions*: Xifeng Yan and Martin Theobald.
- **2008** *Winner*: Ariel Fuxman. *Honorable Mentions*: Cong Yu and Nilesh Dalvi.
- **2009** *Winner*: Daniel Abadi. *Honorable Mentions*: Bee-Chung Chen and Ashwin Machanavajjhala.
- **2010** *Winner:* Christopher Ré. *Honorable Mentions*: Soumyadeb Mitra and Fabian Suchanek.
- **2011** *Winner*: Stratos Idreos. *Honorable Mentions*: Todd Green and Karl Schnaitterz.
- **2012** *Winner*: Ryan Johnson. *Honorable Mention*: Bogdan Alexe.
- **2013** *Winner*: Sudipto Das, *Honorable Mention*: Herodotos Herodotou and Wenchao Zhou.
- **2014** *Winners*: Aditya Parameswaran and Andy Pavlo.
- **2015** *Winner*: Alexander Thomson. *Honorable Mentions*: Marina Drosou and Karthik Ramachandra
- **2016** *Winner*: Paris Koutris. *Honorable Mentions*: Pinar Tozun and Alvin Cheung
- **2017** *Winne*r: Peter Bailis. *Honorable Mention*: Immanuel Trummer
- **2018** *Winne*r: Viktor Leis. *Honorable Mention*: Luis Galárraga and Yongjoo Park

A complete list of all SIGMOD Awards is available at: **https://sigmod.org/sigmod-awards/**

[Last updated : June 30, 2018]

Editor's Notes

Welcome to the June 2018 issue of the ACM SIGMOD Record!

This issue starts with the Database Principles column featuring an article by Michael Benedikt, summarizing recent results on reformulating queries over restricted data interfaces in the presence of integrity constraints. In this setting, the question is how to translate a source query written in some declarative language into a target object, either a query or a program, subject to the requirements that such translation must respect a set of integrity constraints and the interface restriction such as restricted access only to a set of views or a set of access methods. This article presents a common framework for dealing with both types of restricted access, as well as the key ideas behind some of the main techniques.

The Surveys column features two articles. The first article, by Polyzotis et al., discusses data lifecycle challenges in production machine learning. As machine learning has become an essential tool for gleaning knowledge from data, there is a realization that the accuracy of a machine learned model is deeply tied to the data that it is trained on. Drawn from the experience in developing data-centric infrastructure for a production machine learning platform at Google, the authors summarize the interesting research challenges encountered, and survey some of the relevant literature from the data management and machine learning communities. The second article, by Hirzel et al., surveys stream processing languages in the Big Data era. It showcases several languages designed for the purpose of high-volume or scalable data stream processing, articulates their underlying principles, and outlines open challenges.

The Distinguished Profiles column includes two articles. The first article features Kenneth Ross, Professor at Columbia University. In this interview, Ken talks about his well-known work on the semantics of Datalog and main memory databases, as well as his recent work on biology. Ken also shares his experience in transitioning from theoretical topics to systems research, and from computer science to biology and bioinformatics. The second article features Paris Koutris, who won the 2016 ACM SIGMOD Jim Gray Dissertation Award for his thesis entitled "Query Processing in Massively Parallel Systems." Paris is now a professor at the University of Wisconsin-Madison, and he did his Ph.D. work with Dan Suciu at the University of Washington.

The Industry Perspectives column features an article by Michels et al. on the new and improved SQL:2016 standard. This new standard is expanding the SQL language to support new data storage and retrieval paradigms that are emerging from the NoSQL and Big Data worlds. The major new features in SQL:2016 include: (a) support for Java Script Object Notation (JSON) data; (b) polymorphic table functions; and (c) row pattern recognition.

The Research Centers Column features two articles. The first article presents the complex event recognition (CER) group affiliated with the National Centre of Scientific Research "Demokritos" in Greece. The CER group works towards advanced and efficient methods for the recognition of complex events in a multitude of large, heterogeneous data streams, covering topics such as efficient detection of patterns, handling uncertainty and noise in streams, machine learning techniques for inferring interesting patterns, and event forecasting. The second article features the data and information management research team at Nanyang Technological University. It presents an overview of the key research themes in the group including graph data management, social data management, information privacy, and fair peer review management.

Finally, the issue closes with a message from the Editor-in-Chief of ACM TODS on recent changes to the editorial board.

On behalf of the SIGMOD Record Editorial board, I hope that you enjoy reading the June 2018 issue of the SIGMOD Record!

Your submissions to the SIGMOD Record are welcome via the submission site:
http://sigmod.hosting.acm.org/record

Prior to submission, please read the Editorial Policy on the SIGMOD Record's website:
https://sigmodrecord.org

Yanlei Diao

June 2018

Past SIGMOD Record Editors:

Ioana Manolescu (2009-2013)
Ling Liu (2000–2004)
Arie Segev (1989–1995)
Thomas J. Cook (1981–1983)
Daniel O'Connell (1971–1973)
Alexandros Labrinidis (2007–2009)
Michael Franklin (1996–2000)
Margaret H. Dunham (1986–1988)
Douglas S. Kerr (1976-1978)
Harrison R. Morse (1969)
Mario Nascimento (2005–2007)
Jennifer Widom (1995–1996)
Jon D. Clark (1984–1985)
Randall Rustin (1974-1975)

Logic-based Perspectives on Query Reformulation over Restricted Interfaces

Michael Benedikt
University of Oxford, UK
michael.benedikt@cs.ox.ac.uk

ABSTRACT

We overview recent developments on query reformulation over a restricted interface, in the presence of integrity constraints. We overview an approach to the problem via reduction to query containment with constraints, where the reduction makes use of interpolation algorithms from logic. We first present the approach in the context of reformulating one query as another query using a fixed set of tables. We then generalize to reformulation of a query as a plan over a set of access methods.

1. INTRODUCTION

This article summarizes a series of articles [2, 8, 9, 5, 6, 4] revisiting *reformulating queries over restricted data interfaces in the presence of integrity constraints*. In this problem, we start with a *source query* Q written in some declarative language and a set of integrity constraints Σ. We also have some "interface restriction", representing a limit on how data is accessed. We want to translate Q into a target object P — either a query or a program — that satisfies two properties:

- P is equivalent to Q for all query inputs satisfying the constraints Σ
- P satisfies the interface restriction.

We consider two flavors of interface restriction. The first is *vocabulary-based restriction*. We have a subset $\mathcal{V}$ of the tables in the schema, and we want P to be a query referencing only tables in $\mathcal{V}$. The prototypical case is where $\mathcal{V}$ is a set of view tables, and Σ includes the assertion that each view table stores exactly the tuples satisfying the corresponding view definition.

A second kind of interface restriction is given by *access methods*: each table T with n attributes is associated with a set (possibly empty) of access methods. Each method is further associated with a subset of the attributes of T – the input positions. The idea is that a method gives functional access to table T: given a binding for the input positions, it returns the matching tuples in T. Our reformulation problem is to see if Q is equivalent, modulo constraints Σ, to something "executable with respect to the access methods". That is, we want a plan that makes use of the access methods, whose result agrees with Q for all inputs satisfying Σ.

Thus our reformulation problem generalizes both rewriting queries with respect to views, which has been studied for decades, as well as prior work on determining whether a query can be executed with access methods [20, 19, 31, 23, 24, 12].

We present a common framework for dealing with both of these problems, using a reduction to query containment with constraints. From a proof of a query containment one can extract a reformulation, using a technique from logic called *interpolation*. This framework provides new reformulation algorithms both when the target is a query and when the target is a plan. It also gives a common way to see many prior results in the area. For instance, it allows us to re-derive classic results on querying over views, such as Levy, Mendelzon, Sagiv, and Srivastava's [17], and methods for reformulating queries using constraints, such as the Chase and Backchase of Deutsch, Popa, and Tannen [16, 14, 26].

Although the unified presentation of reformulation comes from our own work, it builds on a long line of prior papers. Particularly important is Nash, Segoufin and Vianu's work on characterizing when a query can be expressed using views [25]. Two other key antecedents are Deutsch, Ludäscher, and Nash's paper [12] on querying with access methods and integrity constraints, and the book of Toman and Weddell [30] on reformulation over constraints.

In this survey article we will go through some of the main ideas, skipping most of the details. A full exposition of the techniques, as well as a detailed discussion of related work, can be found in [4].

Organization:. Section 2 contains standard DB preliminaries, as well as some results on query containment with constraints. Section 3 looks at the reformulation problem for vocabulary-based inter-

faces. Section 4 turns to interfaces based on access methods. We present a discussion of implications and future directions in Section 5, before concluding in Section 6.

2. PRELIMINARIES

Data and queries. The basic data model of a querying scenario is given by a *relational schema* $\mathcal{S}$ that consists of a set of *relations* each with an associated *arity* (a positive integer). The *positions* of a relation R of $\mathcal{S}$ are $1, \ldots, n$ where n is the arity of R. An *instance* of R is a set of n-tuples (finite or infinite), and an *instance* I of $\mathcal{S}$ consists of instances for each relation of $\mathcal{S}$. For an instance I of $\mathcal{S}$ and a relation $R \in \mathcal{S}$, the set of tuples assigned to R in I is the *interpretation of* R *in* I. We can equivalently see I as a set of *facts* $R(a_1 \ldots a_n)$ for each tuple $(a_1 \ldots a_n)$ in the instance of each relation R. The *active domain* of I, denoted $\mathsf{adom}(I)$, is the set of all the values that occur in facts of I.

The source queries that are being reformulated will be *conjunctive queries* (CQs) which are expressions of the form $\exists x_1 \ldots x_k \ (A_1 \wedge \cdots \wedge A_m)$, where the A_i are *relational atoms* of the form $R(x_1 \ldots x_n)$, with R being a relation of arity n and $x_1 \ldots x_n$ being variables or constants. A *union of conjunctive queries* (UCQ) is a disjunction of CQs. The target for reformulation of a CQ will not necessarily be another CQ or even a UCQ. Sometimes it will be a formula of first-order logic (FO). FO is built up from relational atoms and equalities using the boolean operations and quantifiers $\forall$ and $\exists$, where quantifiers always range over the active domain. For an instance I and query Q given by an FO formula, the set of bindings for the variables that satisfy the formula is the *output* of Q on I, denoted $Q(I)$.

Integrity constraints. To express integrity constraints on instances, we will use sentences of FO. Some of the results apply only to *dependencies*, which will be either *tuple-generating dependencies* (TGDs) or *equality-generating dependencies* (EGDs).

A TGD is an FO sentence τ of the form $\forall \vec{x} \ (\varphi(\vec{x}) \rightarrow \exists \vec{y} \ \psi(\vec{x}, \vec{y}))$ where φ and ψ are CQs. An EGD is of the form: $\forall \vec{x} \ (\varphi(\vec{x}) \rightarrow x_i = x_j)$ where φ is a CQ whose variables include x_i and x_j.

For brevity, in the sequel, we will omit outermost universal quantifications in dependencies.

Query containment problems. A *query containment with constraints* is an assertion

$$Q \subseteq_\Sigma Q'$$

where Q and Q' are queries given by logical formulas, and Σ is a set of integrity constraints (given by logical sentences). Such a containment *holds* if for every instance I satisfying Σ, the result of Q on I is contained in the result of Q' on I [1]. We say that "Q is contained in Q' with respect to Σ". To verify a query containment, it is necessary and sufficient to find a *proof* in some suitable proof system. There are many proof systems for first-order logic. One example is the *tableau proof system*. A tableau proof witnesses that a first-order logic formula φ is unsatisfiable. It is a tree where every node p is associated with a set of formulas F_p. The root of the tree is associated with the singleton set of formulas $\{\varphi\}$ and every leaf must be associated to a set containing an explicitly contradictory formula (False). A non-leaf node p, associated with formulas F_p, has at most two children. For each child c, the set of formulas F_c is related to F_p by adding on some subformula of a formula $\gamma_p \in F_p$. For example, if F_p includes a formula γ_p that is a disjunction $\gamma_1 \vee \gamma_2$, then one of the children will contain γ_1 and the other will contain γ_2. Tableau proofs give a complete method for detecting unsatisfiability of a sentence:

Proposition 1: If φ is unsatisfiable there is some tableau proof witnessing this. ■

A query containment $Q_1 \subseteq_\Sigma Q_2$ holds exactly when $Q_1 \wedge \Sigma \wedge \neg Q_2$ is unsatisfiable. Thus tableau proofs provide a complete method to verify query containment with arbitrary first-order integrity constraints Σ. Other proof systems for first-order logic include resolution and natural deduction. For query containment problems in which the constraints Σ consist of *dependencies*, there are more specialized proof systems, such as *the chase* [22].

3. REFORMULATING OVER A SUBSET OF THE RELATIONS

We revisit the reduction from reformulation to query containment with constraints implicit in the work of Nash, Segoufin, and Vianu [25]. We will phrase their work in terms of reformulation of a query defined over a *source* vocabulary, where the goal is to translate it into a query over a *target* vocabulary. We have integrity constraints that can involve relations in both the source vocabulary and the target vocabulary. Thus the "interface" is defined by giving the target vocabulary.

Let $\mathcal{S}$ be a collection of relations, Σ a set of integrity constraints, and $\mathcal{V}$ a subset of $\mathcal{S}$. A *first-order reformulation of* Q *over* $\mathcal{V}$ *with respect to* Σ

[1] Note that in this work we will always consider containments over all instances, not just finite ones. The theorems presented here do not hold for general first-order logic if only finite instances are considered. However, for many classes of interest, the results hold verbatim over finite instances [4].

means a safe first-order query $Q_{\mathcal{V}}$ (that is, a query equivalent to a relational algebra query), using only the relations in $\mathcal{V}$, and such that for every instance I satisfying Σ, $Q(I) = Q_{\mathcal{V}}(I)$.

Example 3.1. A university database has a relation Prof containing ids and last names of professors, along with the name of the professor's department. It also has a relation Stud listing the id and last name of each student, along with their advisor's id.

The database does not allow users to access the Prof and Stud relations directly, but instead exposes a view VProf where the id attribute is dropped, and a relation VStud where the advisor's id is replaced with the advisor's last name.

That is, VProf is a view defined by the formula:

$$\{ \mathsf{lname},\ \mathsf{dname} \mid \exists\ \mathsf{profid}\ \mathsf{Prof}(\mathsf{profid},\ \mathsf{lname},\ \mathsf{dname})\}$$

or equivalently by the constraint:

$$(\exists \mathsf{profid}\ \mathsf{Prof}(\mathsf{profid}, \mathsf{lname}, \mathsf{dname})) \leftrightarrow \mathsf{VProf}(\mathsf{lname}, \mathsf{dname})$$

VStud is a view defined by the formula:

$$\{\mathsf{studid}, \mathsf{lname}, \mathsf{profname} \mid \exists \mathsf{profid}\ \exists \mathsf{dname}\ \mathsf{Stud}(\mathsf{studid}, \mathsf{lname}, \mathsf{profid}) \wedge \mathsf{Prof}(\mathsf{profid}, \mathsf{profname}, \mathsf{dname})\}$$

or equivalently by the constraint:

$$[(\exists \mathsf{profid}\ \exists \mathsf{dname}\ \mathsf{Prof}(\mathsf{profid}, \mathsf{profname}, \mathsf{dname}) \wedge \mathsf{Stud}(\mathsf{studid}, \mathsf{lname}, \mathsf{profid})) \leftrightarrow \mathsf{VStud}(\mathsf{studid}, \mathsf{lname}, \mathsf{profname})]$$

Consider the query asking for the names of the advisors of a given student. We can reformulate this query over the VStud view. The reformulation is just the query returning the profname attribute of the view. But a query asking for the last names of all students that have an advisor in the history department can *not* be refomulated using these two views: knowing the advisor's name is not enough to identify the department. ◁

If $\mathcal{L}$ is some subset of FO, we can similarly talk about an $\mathcal{L}$ reformulation of Q over $\mathcal{V}$ with respect to Σ.

One of the main ideas of [25] is a reduction of finding a reformulation to verifying a query containment with constraints, and the key to this reduction is the notion of *determinacy*, which we define next. If Σ is a collection of first-order constraints, we say that a first-order query Q over $\mathcal{S}$ is *determined over* $\mathcal{V}$ *relative to* Σ if:

> For any two instances I and I' that satisfy Σ and have the same interpretation of all relations in $\mathcal{V}$ (that is, they have the same T-facts for each $T \in \mathcal{V}$), we have $Q(I) = Q(I')$.

That is, two instances that agree on $\mathcal{V}$ must agree on Q. How does determinacy connect with reformulation? We first show that determinacy boils down to checking query containment with constraints. Let us extend our original schema for the constraints Σ and the query Q by making a copy R' of every relation R in the schema. Let Q' be the copy of Q on the new relations, and Σ' be the copy of the constraints Σ on the new relations. Our assumption of determinacy of Q can be restated as a query containment problem $Q \subseteq_\Gamma Q'$, where Γ contains Σ, Σ' and the additional *interface axioms*:

$$\bigwedge_{T \in \mathcal{V}} \forall \vec{y}\ T(\vec{y}) \leftrightarrow T'(\vec{y})$$

This is the *query containment corresponding to determinacy of* Q *over* $\mathcal{V}$ *relative to* Σ.

We now want to argue that when the query containment above holds, we can get a reformulation of Q over $\mathcal{V}$ with respect to Σ. To do this, we need to bring into the picture *interpolation*. If we have a query containment problem $Q \subseteq_\Gamma Q'$, and a partition of Γ into Γ_1 and Γ_2, an *interpolant* for the containment and partition is a formula χ such that:

- Q is contained in χ with respect to Γ_1 and χ is contained in Q' with respect to Γ_2
- Every relation in χ occurs in both $\{Q\} \cup \Gamma_1$ and in $\{Q'\} \cup \Gamma_2$.

The crucial facts about interpolants that are relevant to us are:

- *For every query containment problem* $Q \subseteq_\Gamma Q'$*, for every partition of* Γ *into* Γ_1, Γ_2*, there is an interpolant.* Thus in particular if Q is determined over $\mathcal{V}$ relative to Σ, then for every partition of Γ the containment $Q \subseteq_\Gamma Q'$ above corresponding to determinacy has an interpolant.
- Suppose that the query containment corresponding to determinacy of Q over $\mathcal{V}$ relative to Σ holds, and partition Γ above into $\Gamma_1 = \Sigma$, $\Gamma_2 = \Sigma' \cup$ the interface axioms. Let $Q_{\mathcal{V}}$ be any interpolant for the query containment relative to this partition. Then $Q_{\mathcal{V}}$ is a reformulation of Q over $\mathcal{V}$ with respect to Σ.

The first item, saying that interpolants exist, is a basic result in logic; we do not prove it here. We give the short proof of the second item, saying that

interpolants for this particular partition give reformulations. It is a variant of an argument in [11].

PROOF. For simplicity, in the proof we assume Q is a sentence. The definition of interpolant says that $Q_{\mathcal{V}}$ can use only relations that occur both in $\{Q\} \cup \Gamma_1$ and also in $\Gamma_2 \cup \{Q'\}$. This implies that $Q_{\mathcal{V}}$ uses only relations in $\mathcal{V}$.

Since $Q \subseteq_{\Gamma_1} Q_{\mathcal{V}}$, we know that if an instance satisfies the constraints Σ and also satisfies Q, it must also satisfy $Q_{\mathcal{V}}$.

We argue that if an instance satisfies Σ and also satisfies $Q_{\mathcal{V}}$, it must also satisfy Q. Fix I satisfying Σ such that $Q_{\mathcal{V}}$ holds in I. Extend I to an instance $I + I'$ by letting the interpretation of each primed relation R' be the same as the corresponding unprimed relation R of I. The instance $I + I'$ satisfies $Q_{\mathcal{V}}$, Σ', and the interface axioms. Since $Q_{\mathcal{V}} \subseteq_{\Gamma_2} Q$, we know that $I + I'$ satisfies Q'. By the construction of $I + I'$, this means I satisfies Q.

We have shown that for an instance satisfying Σ, Q holds if and only if $Q_{\mathcal{V}}$ holds, which means $Q_{\mathcal{V}}$ is a reformulation.

Since the existence of a reformulation implies that the query containment for determinacy holds, we have the following result:

Theorem 3.1: A conjunctive query Q has a first-order reformulation with respect to vocabulary $\mathcal{V}$ and constraints Σ if and only if Q is determined over $\mathcal{V}$ relative to Σ if and only if the query containment corresponding to determinacy of Q over $\mathcal{V}$ relative to Σ holds. ■

Finding the reformulation. Theorem 3.1 reduces the problem of *existence of a reformulation* to a query containment problem. But of course, we do not just want to know if a reformulation exists, we want to be able to find it. There is a refinement of Theorem 3.1 that talks about finding the reformulation from a witness that the query containment holds. The witness we require is a *proof*. We mentioned in the preliminaries that there are many proof systems for first-order logic, and most of them admit feasible interpolation algorithms. One example is the tableau proof system mentioned in the preliminaries. One can find interpolants quickly from tableau proofs:

Proposition 2: There is a polynomial time algorithm that given a tableau proof that $Q_1 \subseteq_\Sigma Q_2$ and a partition of Σ into Σ_1, Σ_2, finds an interpolant χ for the containment and partition. ■

Using Proposition 2 we can get the refined version of Theorem 3.1 mentioned above:

Theorem 3.2: A conjunctive query Q has a first-order reformulation with respect to vocabulary $\mathcal{V}$ and constraints Σ if and only if the query containment for determinacy of Q over $\mathcal{V}$ holds. Further, given a tableau proof of the query containment we can extract a reformulation in polynomial time. ■

If the constraints Σ are arbitrary first-order sentences, we cannot bound the time taken to find tableau proof, since query containment with first-order constraints is undecidable. But for many restricted classes of constraints – e.g. referential constraints – we can show that the query containment for determinacy is also decidable. Indeed, for many classes C of constraints the query containment problem for determinacy is no more complex than the problem of query containment for the class C.

There is a subtlety we should mention. In our reformulation problems we start with a CQ, and we are interested in reformulations that can be converted to relational algebra. That is, we want reformulations that are not arbitrary first-order, but formulas which only quantify over the active-domain, and where the free variables are safe. Using classical tableau with prior interpolation procedures does not give us this. But by varying both the proof system and the interpolation algorithm slightly, we can get active-domain formulas. Safe reformulations can be achieved by post-processing the interpolants. Details can be found in [4].

3.1 Variation: vocabulary-based reformulation with positive existential queries

We now explore what happens when we restrict the target language for a reformulation. A *positive existential formula with inequalities* ($\exists^{+,\neq}$ formula) is a formula built up using only existential quantification, starting from atomic relations and inequalities. By convention, we also consider the formula False to be positive existential with inequalities. A safe FO formula in this class is equivalent to a relational algebra expression that does not have the difference operator, but allows inequalities in selections. Thus we will sometimes refer to $\exists^{+,\neq}$ formulas as "$USPJ^{\neq}$ queries".

Given a conjunctive query Q, restricted vocabulary $\mathcal{V}$, and constraints Σ given by FO sentences, we are interested in getting a $\exists^{+,\neq}$ reformulation of Q over $\mathcal{V}$ with respect to Σ. This means we want a $\exists^{+,\neq}$ formula over $\mathcal{V}$ that agrees with Q for instances satisfying the constraints.

The query containment for $\exists^{+,\neq}$ reformulation. We start by finding the appropriate variant of determinacy equivalent to a query Q having a $\exists^{+,\neq}$ reformulation. The property was isolated in

[25]. We say that a query Q over schema Sch is *monotonically-determined over* $\mathcal{V}$ relative to Σ if:

> whenever we have two instances I, I' that satisfy Σ and for each relation $T \in \mathcal{V}$, the interpretation of $T \in I$ is a subset of the interpretation of T in I', then $Q(I) \subseteq Q(I')$.

If a $\exists^{+,\neq}$ formula $Q_{\mathcal{V}}$ over $\mathcal{V}$ is true on an instance I, then it is true on any instance I' which only adds tuples to the relations in $\mathcal{V}$. It follows that monotonic-determinacy of Q over $\mathcal{V}$ relative to Σ is a *necessary* condition for Q to have a $\exists^{+,\neq}$ reformulation over $\mathcal{V}$ with respect to Σ.

We can express monotonic-determinacy as a query containment problem. Again we will use a vocabulary that allows us to talk about two copies of the relations, with R' being a copy of R. We let Σ' be a copy of the constraints Σ where each occurrence of a relation R in $\mathcal{S}$ has been replaced by a copy R'. And we let Q' be defined from Q analogously.

Then monotonic-determinacy of a first-order query Q over $\mathcal{V}$ relative to Σ can be restated as saying that the query containment $Q \subseteq_\Gamma Q'$ holds, where Γ contains Σ, Σ' and the "forward interface axiom":

$$\bigwedge_{T \in \mathcal{V}} \forall \vec{y}\ T(\vec{y}) \rightarrow T'(\vec{y}))$$

That is, the difference from determinacy is that we have only implication in the "forward" direction, from unprimed to primed, while for determinacy we have implications in both directions. This is the *query containment for monotonic-determinacy.*

Generating $\exists^{+,\neq}$ reformulations from proofs of the query containment. We now give a result saying that from proofs of the query containment for monotonic-determinacy, we get $\exists^{+,\neq}$ reformulations. It is an analog of Theorem 3.1.

Theorem 3.3: If the constraints Σ are first-order then conjunctive query Q has a $USPJ^{\neq}$ reformulation over $\mathcal{V}$ relative to Σ if and only if the query containment for monotonic-determinacy holds. ■

Theorem 3.3 is proven using the same technique as Theorem 3.1: applying an interpolation algorithm to the query containment associated to monotonic-determinacy. One needs to ensure that the interpolants have some additional properties in order to be sure that the interpolant coming from the proof does not have negation. Many interpolation algorithms are known to ensure this additional property [21]. As with Theorem 3.1, there is a variant that tells us we can find the reformulation effectively given a proof of the query containment.

3.2 Variation: existential reformulation

We now look at another variation of the reformulation problem: determining whether a query can be reformulated using an *existential formula*, or equivalently a UCQ with negation allowed only on atomic formulas. That is, a formula that is built up from atoms *and* negated atoms by positive boolean operators and existential quantification. There are conjunctive queries that are equivalent to existential formulas but not to positive existential ones. For example, in the absence of any constraints $\exists x\ S(x) \wedge \neg R(x)$ is not equivalent to a positive existential formula.

As before, let Sch be a schema with a set of integrity constraints Σ in FO, and $\mathcal{V}$ a subset of the relations of Sch. We start by isolating a determinacy property that Q must have in order to possess an existential reformulation.

For a set of relations $\mathcal{V}$ and instances I and I', we say that I *is a* $\mathcal{V}$ *induced-subinstance of* I' provided for each $T \in \mathcal{V}$ two conditions hold. First, I' contains every fact $T(\vec{c})$ in I. Second, I contains every fact $T(c_1 \ldots c_n)$ in I' such that each c_i occurs in the domain of some relation of $\mathcal{V}$ in I. We say that a query Q over schema Sch is *induced-subinstance-monotonically-determined over* $\mathcal{V}$ relative to Σ if:

Whenever we have two instances I, I' that satisfy Σ, and I is a $\mathcal{V}$ induced-subinstance of I', then $Q(I) \subseteq Q(I')$.

If an existential formula over $\mathcal{V}$ is true on an instance I, then it is true on any instance I' which only adds tuples to the relations in $\mathcal{V}$ and never "destroys a negated assertion about a relation of $\mathcal{V}$ holding in I". From this we see that if a formula is equivalent to an existential formula under constraints Σ, then the formula is induced-subinstance-monotonically-determined over $\mathcal{V}$ relative to Σ.

As in the previous cases, we can translate this property into a query containment with constraints, but it will be slightly more complicated than determinacy or monotonic-determinacy. Let $\mathsf{InDomain}_{\mathcal{V}}(x)$ abbreviate the formula:

$$\bigvee_{T \in \mathcal{V}} \bigvee_{j} \exists w_1 \ldots \exists w_{j-1}\ \exists w_{j+1} \ldots w_{\mathsf{arity}(T)}\ T(w_1, \ldots w_{j-1}, x, w_{j+1}, \ldots, w_{\mathsf{arity}(T)})$$

So $\mathsf{InDomain}_{\mathcal{V}}$ states that x is in the domain of a relation in $\mathcal{V}$. The *query containment for induced-subinstance-monotonic-determinacy* is $Q \subseteq_\Gamma Q'$, where Γ contains Σ, Σ', and also the following two addi-

tional axioms:

$$\bigwedge_{T \in \mathcal{V}} (\forall \vec{y}\ T(\vec{y}) \rightarrow T'(\vec{y}))$$

$$\bigwedge_{T \in \mathcal{V}} (\forall \vec{y}\ \bigwedge_{i} \mathsf{InDomain}_{\mathcal{V}}(y_i) \wedge T'(\vec{y}) \rightarrow T(\vec{y})\ )$$

Comparing with the two previous query containments, we have the forward interface axiom, used in the axioms for determinacy and monotone-determinacy. We also have a restriction of the backward interface axiom used in determinacy. It is easy to see that induced-subinstance-monotonic-determinacy is equivalent to this containment holding.

Extracting reformulations from a proof of the query containment. Continuing the prior pattern, we can show that from a proof of the query containment corresponding to induced-subinstance-monotonic-determinacy, we can extract an existential reformulation from it:

Theorem 3.4: If the constraints Σ are in FO, and CQ Q is induced-subinstance-monotonically-determined over $\mathcal{V}$ relative to Σ, then there is an existential formula $\varphi(\vec{x})$ using only relations in $\mathcal{V}$ that is a reformulation of Q with respect to Σ. ■

The bottom line on vocabulary-based reformulation is:

For every target language, we have a different query containment problem. From proofs of the query containment, we can extract reformulations.

When constraints are dependencies, one can use chase proofs to verify the query containment. For chase proofs, extraction of the reformulation from a proof turns out to be very simple (see [9, 4]).

4. ACCESS METHODS

In the previous section the target of reformulation was specified through vocabulary restrictions. We wanted a query that used a fixed set of target relations, perhaps restricted to be positive existential or existential. In this section we deal with a finer notion of reformulation, where the target has to satisfy *access restrictions*.

Access methods are close to the traditional notion of interface in programming languages: a set of functions that access the data. A specification of this interface will be an extended set of metadata describing both the format of the data (e.g. the vocabulary that would be used in queries and constraints) and the access methods (functions that interact with the stored data).

An *access schema* consists of:

- A collection of relations, each of a given arity.
- A finite collection C of schema constants ("Smith", 3, ...). Schema constants represent a fixed set of values that will be known to a user prior to interacting with the data. Values that can be used in queries and constraints should be schema constants, as before. In addition, any fixed values that might be used in plans that implement queries should come from the set of schema constants. For example, a plan that reformulates a query about the mathematics department might involve first putting the string "mathematics" into a directory service.
- For each relation R, a collection (possibly empty) of *access methods*[2]. Each access method mt is associated with a collection (possibly empty) of positions of R – the *input positions* of mt.
- Integrity constraints, which are sentences of first-order logic as before.

Example 4.1. Suppose we have a Profinfo relation containing information about faculty, including their last names, office number, and employee id. We have a restricted interface that requires giving an employee id as an input.

Intuitively, in such a schema we can not find out information about all professors. But if we had a query asking about a particular professor, hardcoding the professor's employee id, we would be able to use the interface to answer it. ◁

An *access* (relative to a schema as above) consists of an access method of the schema and a *method binding* — a function assigning values to every input position of the method. If mt is an access method on relation R with arity n, I is an instance for a schema that includes R, and AccBind is a method binding on mt, then the *output* or *result* of the access (mt, AccBind) on I is the set of n-tuples $\vec{t}$ such that $R(\vec{t})$ holds in I and $\vec{t}$ restricted to the input positions of mt is equal to AccBind.

An access method may be "input-free": have an empty collection of input positions. In this case, the only access that can be performed using the method is with the empty method binding.

The goal is to reformulate source queries in a target language that represents the kind of restricted computation done over an interface given by an access schema. We formalize this as a language of *plans*. Plans are straight-line programs that can perform accesses and manipulate the results of accesses using relational algebra operators. This lan-

[2]Our definition of "access methods" is a variant of the terminology "access patterns" or "binding patterns" found in the database literature.

guage could model, at a high-level, the plans used internally in a database management system. It could also describe the computation done within a data integration system, which might access remote data via a web form or web service and then combine data from different sources using SQL within its own database management system.

Example 4.2. Suppose we have a Profinfo relation with a restricted interface that requires giving an employee id as an input, as in Example 4.1. But we also have a Udirectory relation containing the employee id and last name of every university employee, with an input-free access method. The fact that the directory contains every employee and that a professor is an employee is captured by the integrity constraint stating that every employee id in the Profinfo is also contained in Udirectory.

Suppose we are interested in the query asking for ids of faculty named "Smith":

$$Q = \exists \text{onum } \mathsf{Profinfo}(\mathsf{eid}, \mathsf{onum}, \text{"Smith"})$$

A reformulation is a program using the given methods, where the program is equivalent to Q for all inputs satisfying the integrity constraints Σ.

One can easily see that there is a reformulation of Q using these access methods: we simply access Udirectory to get all the employee ids, then use these to access Profinfo, filtering the resulting tuples to return only those that have name "Smith".

On the other hand, if we did not have access to Udirectory, we can see intuitively that there is no such reformulation. ◁

Formally, we have a plan language with two basic commands. The first is an *access command*. Over a schema Sch with access methods, an access command is of the form:

$$T \Leftarrow_{\mathsf{OutMap}} \mathsf{mt} \Leftarrow_{\mathsf{InMap}} E$$

where:

- E is a relational algebra expression, the *input expression*, over some set of relations not in Sch (henceforward "temporary relations");
- mt is a method from Sch on some relation R;
- InMap, the *input mapping* of the command, is a function from the output attributes of E onto the input positions of mt;
- T, the *output relation* of the command, is a temporary relation;
- OutMap, the *output mapping* of the command, is a bijection from positions of R to attributes of T.

Note that an access command using an input-free method must take the empty relation algebra expression $\emptyset$ as input.

The manipulation of data retrieved by an access is modeled with the other primitive of our plan language, a *middleware query command*. These are of the form $T := Q$, where Q is a relational algebra expression over temporary relations and T is a temporary relation. We use the qualifier "middleware" to emphasize that the queries are performed on temporary relations created by other commands, rather than on relations of the input schema.

A *relational algebra-plan* (or simply, *RA-plan*) consists of a sequence of access and middleware query commands, ending with at most one *return command* of the form Return E, where E is a relational algebra expression.

Example 4.3. We return to Example 4.2 where we had two sources of information. One was Profinfo, which was available through an access method $\mathsf{mt}_{\mathsf{Profinfo}}$ requiring input on the first position. The second was Udirectory, which had an access method $\mathsf{mt}_{\mathsf{Udirectory}}$ requiring no input. Our query Q asked for ids of faculty named "Smith". One plan that is equivalent to Q would be represented as follows

$$\begin{aligned}
&T_1 \Leftarrow \mathsf{mt}_{\mathsf{Udirectory}} \Leftarrow \emptyset \\
&T_2 := \pi_{\mathsf{eid}}(\sigma_{\mathsf{lname}=\text{"Smith"}} T_1) \\
&T_3 \Leftarrow \mathsf{mt}_{\mathsf{Profinfo}} \Leftarrow T_2 \\
&\mathsf{Return}\ \pi_{\mathsf{eid}}(T_3)
\end{aligned}$$

Above we have omitted the mappings in writing access commands, since they can be inferred from the context. ◁

Fragments of the plan language. There are fragments of our plan language, analogs of the standard fragments of relational algebra and first-order logic. In RA-plans, we allowed arbitrary relational algebra expressions in both the inputs to access commands and the middleware query commands. We can similarly talk about $USPJ^{\neq}$*-plans*, where both kinds of commands can only use $USPJ^{\neq}$ queries.

Plans that reformulate queries. We now define what it means for a plan to correctly implement a query. Given an access schema Sch, a plan *reformulates* a query Q with respect to Sch if for every instance I satisfying the constraints of Sch, the output of the plan on I is the same as the output of Q. We often omit the schema from our notation, since it is usually clear from context, saying that a plan PL reformulates Q. Note that this extends the notion of a query $Q_{\mathcal{V}}$ over relations $\mathcal{V}$ reformulating a query Q.

4.1 Reduction to query containment

Recall from Section 3 that a query Q had a reformulation with respect to a vocabulary-based in-

terface $\mathcal{V}$ if and only if the output of Q was determined by the data stored in $\mathcal{V}$. In the case of access methods, we would like to say that Q can be reformulatable using the access methods if and only if it is "determined by the data we can get via the access methods". We will require some auxiliary definitions to formalize what we mean by "the data we can get via the access methods".

Given an instance I for schema Sch the *accessible part of* I, denoted AccPart(I) consists of all the facts over I that can be obtained by starting with empty relations and iteratively entering values into the access methods. This will be an instance containing a set of facts Accessed$R(v_1 \dots v_n)$, where R is a relation and $v_1 \dots v_n$ are a subset of the values in the domain of I such that $R(v_1 \dots v_n)$ holds in I. The content of relations Accessed$R$ will be formed as a limit of inductively-defined sets AccessedR_i. In the inductive process we will also build a set of elements accessible$_i$. If Sch contains no schema constants, we start the induction with relations AccessedR_0 and accessible$_0$ empty. We then iterate the following process until a fixpoint is reached:

$$\text{accessible}_{i+1} = \text{accessible}_i \cup \bigcup_{\substack{R \text{ a relation} \\ j \leq \text{arity}(R)}} \pi_j(\text{Accessed}R_i)$$

and

$$\text{Accessed}R_{i+1} = \text{Accessed}R_i \cup \bigcup_{\substack{(R,\{j_1,\dots,j_m\}) \\ \text{there is a method on } R \text{ with inputs } j_1,\dots,j_m}} \{v_1 \dots v_n | R(v_1 \dots v_n) \text{ in } I,\ v_{j_1} \dots v_{j_m} \in \text{accessible}_i\}$$

Above $\pi_j(\text{Accessed}R_i)$ denotes the projection of AccessedR_i on the j^{th} position. For a finite instance, this induction will reach a fixpoint after $|I|$ iterations, where $|I|$ denotes the number of facts in I. For an arbitrary instance the union of these instances over all i will be a fixpoint.

Assuming Sch does include schema constants, we modify the definition by starting with accessible$_0$ consisting of the schema constants, rather than being empty.

Above we consider AccPart(I) as a database instance for the schema with relations accessible and Accessed$R$. Below we will sometimes refer to the values in the relation accessible as the *accessible values of* I.

Example 4.4. Suppose our schema has a relation Related of arity 2, with an access method $\text{mt}_{\text{Related}}$ with input on the first position of Related. The schema has exactly one schema constant "Jones". Let instance I consist of facts

$$\{\text{Related}(\text{“Jones”}, \text{“Kennedy”}), \text{Related}(\text{“Kennedy”}, \text{“Evans”}), \text{Related}(\text{“Smith”}, \text{“Thompson”})\}$$

We construct the accessible part of I. We begin by computing:

$$\text{AccessedRelated}_0 = \emptyset, \quad \text{accessible}_0 = \{\text{“Jones”}\}$$

That is, initially the accessible part contains no facts and the only accessible constant is the schema constant "Jones".

We can now apply the inductive rules to get after one iteration:

$$\text{AccessedRelated}_1 = \{(\text{“Jones”}, \text{“Kennedy”})\}$$
$$\text{accessible}_1 = \{\text{“Jones”}, \text{“Kennedy”}\}.$$

and after a second iteration:

$$\text{AccessedRelated}_2 = \{(\text{“Jones”}, \text{“Kennedy”}), (\text{“Kennedy”}, \text{“Evans”})\}$$
$$\text{accessible}_2 = \{\text{“Jones”}, \text{“Kennedy”}, \text{“Evans”}\}$$

At this point, we have reached a fixpoint, so the accessible part of I consists of facts

$$\{\text{AccessedRelated}(\text{“Jones”}, \text{“Kennedy”}), \text{AccessedRelated}(\text{“Kennedy”}, \text{“Evans”})\}$$

The accessible values of I are

$$\{\text{“Jones”}, \text{“Kennedy”}, \text{“Evans”}\}$$

◁

Query Q is said to be *access-determined* over Sch if for all instances I and I' satisfying the constraints of Sch with AccPart(I) = AccPart(I') we have $Q(I) = Q(I')$. If a query is *not* access-determined, it is obvious that it cannot be reformulated through any plan, since any plan can only read tuples in the accessible part.

Example 4.5. We return to the setting of Example 4.1, where we have a Profinfo relation containing information about faculty, including their last names, office number, and employee id, but with only an access method $\text{mt}_{\text{Profinfo}}$ that requires giving an employee id as an input. We consider again the query Q asking for ids of faculty named "Smith", where "Smith" is a schema constant.

We show that Q is not access-determined. For this, take I to be any instance that contains exactly one tuple, with lastname "Smith", but with an employee id that is not one of the schema constants. Let I' be the empty instance. The accessible parts of I and I' are empty, since in both cases when we enter all the constants we know about in $\text{mt}_{\text{Profinfo}}$,

we get the empty response. But Q has an output on I but no output on I'.

I and I' witness that Q is not access-determined. From this we see that Q can not be reformulated by any plan using only $\mathsf{mt}_{\mathsf{Profinfo}}$. ◁

We now show that access-determinacy reduces to a query containment. Given a schema Sch with constraints Σ and access methods, we form a schema $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ that has only integrity constraints. $\mathsf{AcSch}^{\leftrightarrow}$ will contain two copies of every relation in Sch, with the copy of R denoted as R'. The constraints of $\mathsf{AcSch}^{\leftrightarrow}$ will include all constraints Σ of Sch, a copy Σ' of the constraints on the new relations, and also the following additional axioms, which we call *accessibility axioms*. The first set of axioms, which we call *forward accessibility axioms*, are as follows (universal quantifiers omitted):

$$\bigwedge_{i \leq m} \mathsf{accessible}(x_{j_i}) \wedge R(x_1 \ldots x_n) \rightarrow R'(x_1 \ldots x_n) \wedge \bigwedge_i \mathsf{accessible}(x_i)$$

Above, R is a relation of Sch having an access method with input positions $j_1 \ldots j_m$.

The second set of axioms, *backward accessibility axioms*, just reverses the roles of R and R':

$$\bigwedge_{i \leq m} \mathsf{accessible}(x_{j_i}) \wedge R'(x_1 \ldots x_n) \rightarrow R(x_1 \ldots x_n) \wedge \bigwedge_i \mathsf{accessible}(x_i)$$

where again R is a relation of Sch having an access method with input positions $j_1 \ldots j_m$.

Intuitively, the primed and unprimed copies are a way of writing a statement about two instances. The relation $\mathsf{accessible}$ represents the common accessible values of the two instances. The axioms state that both instances satisfy the constraints, and ensure that their accessible parts are the same. The notation $\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})$ emphasizes that we have constraints from primed to unprimed and vice versa.

As before, we extend the priming notation to queries, letting Q' be obtained from Q by replacing each relation R by R'. The query containment for access-determinacy is then $Q \subseteq_{\mathsf{AcSch}^{\leftrightarrow}(\mathsf{Sch})} Q'$. Analogously to the vocabulary-based case, we can show that this query containment captures the proposed determinacy property, access-determinacy.

We can also show that whenever the query containment for access-determinacy holds, we can extract a plan that reformulates Q:

Theorem 4.1: For any CQ Q and access schema Sch with constraints in FO, the query containment for access-determinacy holds if and only if there is an RA-plan reformulating Q (over instances of Sch). ■

The proof of Theorem 4.1 uses another refinement of interpolation. Theorem 4.1 only talks about discovering whether a plan exists. There is a variation that says we can find the plan given a suitable proof, analogously to the vocabulary-based setting.

4.2 Variation: plans without negation

When we defined the language of RA-plans,we argued that it forms a natural counterpart to relational algebra in the setting where the interface to data is given by a set of access methods. The analog of $\exists^{+,\neq}$ formulas (equivalent to $USPJ^{\neq}$ queries) in the setting of plans are the $USPJ^{\neq}$-plans mentioned earlier, where we do not allow relational algebra difference in any expressions within commands. We will now consider the problem of reformulating a query as a $USPJ^{\neq}$-plan.

We need a variation of determinacy corresponding to a plan that only uses "accessible data" and only uses it monotonically. We say Q is *access-monotonically-determined* over Sch if whenever we have instances I and I' satisfying the constraints of Sch with every fact of $\mathsf{AccPart}(I)$ contained in $\mathsf{AccPart}(I')$, then $Q(I) \subseteq Q(I')$.

The query containment corresponding to access-monotonic-determinacy is simple: we take the same queries Q, Q' as with access-determinacy, but we include only the forward accessibility axioms.

It is easy to verify that the query containment captures access-monotonic-determinacy. And once again, we can take a proof of the query containment and extraction a reformulation, using the appropriate interpolation algorithm:

Theorem 4.2: For any CQ Q and access schema Sch containing constraints specified in FO, there is a $USPJ^{\neq}$-plan reformulating Q (over instances in Sch) if and only if the query containment for access-monotonic-determinacy holds if and only if Q is access-monotonically-determined over Sch. Furthermore, for every tableau proof witnessing the query containment, we can extract a $USPJ^{\neq}$-plan that reformulates Q. ■

4.3 More variations on restrictions based on access methods

Recall that for vocabulary-based restrictions, there was a variation of the technique for existential reformulation: we are looking for a reformulation and

allow it to use negation, but only at the atomic level. There is a similar variation for plans that "only use negation at the atomic level". The definition of such plans is a bit technical, since in the plan language simply restricting query middleware commands to only use atomic negation is not enough. The definitions and the details of the reformulation method can be found in [9, 4].

In our plans we assumed that an access method on relation R returns all the matching tuples on R. Web service access methods may impose *result limits*, setting an upper bound on the number of matching tuples returned. Another variation of the method, described in [1], shows how to find reformulations with access methods that include result bounds.

5. DISCUSSION

We have presented a few theorems that are representative of the reduction to query containment. In this section we go through some of the implications of the results. This will include a discussion of the main theoretical advantage of the technique, the immediate prospects of applying the results in practice, and remarks on the history of the topic.

5.1 Querying over interfaces

Reformulation is a very broad topic, with still many aspects untouched. Dimensions of the problem include:

- *The logical operators allowed in the target of reformulation.* In this article we have looked at three flavors of reformulation depending on the operators allowed. In "first-order" or "relational-algebra" reformulation, negation is allowed in the target. In "monotone" or "positive-existential" reformulation, we want a reformulation that does not use negation. In between is "existential reformulation", in which we allow negation, but not nested. Surely there are many more possibilities for the allowed operators.
- *The notion of interface.* Reformulation is about synthesizing an implementation with a given interface. Here we have dealt with only two, but there are many notions of data interface that can be considered.
- *The class of constraints.* Integrity constraints are implicit in any analysis of reformulation. In the case of reformulation over views, the constraints are just the view definitions. But one can consider much broader or more restricted classes of constraints, and the class considered will impact the algorithms.
- *The reasoning system used to verify that a reformulation exists.* For constraints that are dependencies, the natural reasoning system for proving query containments for reformulation is the chase. But other proof systems can be used even in the case of dependencies, and more general proof systems need to be used once one goes beyond dependencies. We mentioned tableau and resolution as proof systems in some of our results, but there are many proof systems that can be applied.

The majority of prior work has focused on one spot within the space:

- the interface is given by views
- the constraints consist of view definitions and/or weakly-acyclic dependencies
- the target is a monotone query;
- the reasoning system is the chase.

This case is of course important in practice, and it is attractive because it allows intuitive algorithms like the Chase & Backchase (C&B) [16, 14, 26]. The biggest impact of the work presented here is a common framework for exploring a much wider space. This has a conceptual benefit, and provides, at least in principle, reformulation algorithms for classes that have not been considered in the past.

There are many other kinds of data interfaces that could be considered e.g. web interfaces that allow one to send SQL commands; keyword base interfaces. And for other data models there are still further possibilities. The broader framework presented here could be useful for generalizing reformulation to new contexts.

5.2 Practical aspects

The approach based on reduction to query containment has an advantage in its generality. But does it provide better algorithms in practice?

Suppose we specialize this framework to the most well-studied setting: a vocabulary-based interface, the target being $USPJ^{\neq}$ queries ("monotone queries", for short) and constraints that are dependencies where the chase process [22] terminates. If we use the chase as our proof system, and look at the algorithm that results from applying the machinery, what we obtain is a variation of the C&B algorithm mentioned above. The framework by itself only tells us how to find one reformulation. For finding a *good* reformulation, one needs a way of searching through the space of proofs, and then selecting the best one. When constraints are dependencies, there are additional optimizations for efficiently searching the search space, and these have been incorporated into the C&B [16]. In the same way, traditional

algorithms for finding negation-free rewritings over CQ views [18, 27] include important techniques for efficiently enumerating the space of rewritings. One does not get this "for free" from the interpolation framework.

Let us now stick to the vocabulary-based setting, searching for monotone reformulations, but let our constraints be *disjunctive dependencies*, rules with disjunction in the head. One can use an extension of the chase, the "disjunctive chase" [13] as a proof system. Specializing our framework using interpolation to this setting, one gets a variation of the C&B using disjunction [12]. However, there are other proof systems that one can apply, such as tableau proofs or resolution, and applying the framework with these will give different rewritings than those provided by the C&B. Preliminary results [3] show that the interpolation-based approach on top of resolution can give much more succinct reformulations than the approach using the disjunctive chase.

When the constraints go beyond disjunctive dependencies, we know of no competitor to the approach via interpolation. But to make use of the technique here may require more complex theorem proving techniques. Similarly, if we look at finding general first-order reformulations over views, rather than monotone rewritings, we can still apply the technique to reduce to theorem proving, but the theorem proving problem is undecidable in general [15], so we may need to make use of incomplete or non-terminating methods. Further, to find a good reformulation, one needs access to multiple proofs from a theorem prover, and a way to search through these proofs: theorem provers do not have such APIs at present.

One of the simplest practical application of the frameworks is in the case of access methods and integrity constraints in the form of dependencies with terminating chase. In a data integration setting, these constraints may relate local sources that have access methods to an integrated schema. They may also restrict the local sources. Given a query (e.g. over an integrated schema), the access methods, and the constraints, the variant of the approach given in Subsection 4.2 can be applied to determine whether a $USPJ^{\neq}$-plan can be generated, and if so synthesize a plan. We have applied the framework to a number of application settings, ranging from web services [7] to more traditional database access methods [5, 6].

5.3 Algorithms and semantics

Finally, we want to mention a general "lesson learned" from this line of work, concerning the interplay of algorithms and semantics.

There is a long history of algorithmic work for rewriting queries over restricted interfaces. Examples include the work of Levy, Mendelzon, Sagiv, and Srivastava [17], leading to the well known bucket [18] and MiniCon [27] algorithms. The C&B is another example of a clever algorithm for finding reformulations, in the more general setting of queries over a subset of the relations with respect to integrity constraints [16, 14, 26].

A parallel line of research deals with characterizing queries that can be rewritten in certain ways, relating the syntactic restrictions in the target language and semantic properties of the source query. Examples in this line are the homomorphism preservation theorem (see [28]), which states that a first-order formula can be rewritten as a UCQ exactly when it is preserved under homomorphism. In databases, the semantic line includes the work of Segoufin and Vianu [29] and the subsequent TODS paper of Nash, Segoufin, and Vianu [25]. They defined the notion of determinacy we used in Section 3, and showed that it characterizes queries that have relational algebra reformulations.

Another message of this work is that the semantic and algorithmic lines are connected. The semantic approach gives a clean way to see that certain reformulations exists. But it can be converted to an algorithmic technique, applicable not only to view-based reformulation, but to reformulation with integrity constraints and access methods. We think that reformulation gives a nice example of how expressiveness results and algorithmic methods can interact.

6. CONCLUSION

We have presented an overview of a recipe for query reformulation over interfaces. It involves two components: a reduction to query containment problems, and then the use of interpolation algorithms applied to proofs of a containment. We have given an idea of the generality of the framework, showing it is applicable to different kinds of interfaces and different kinds of logical operators in the reformulation target.

A more detailed look at reformulation can be found in Toman and Weddell's book [30], or in the book that takes the perspective presented here, [4]. For the reader interested primarily in the case of TGD constraints, the paper [9] gives a shorter overview.

7. REFERENCES

[1] Antoine Amarilli and Michael Benedikt. When can we answer queries using

result-bounded services. In *PODS*, 2018.
[2] Vince Bárány, Michael Benedikt, and Pierre Bourhis. Access restrictions and integrity constraints revisited. In *ICDT*, 2013.
[3] Michael Benedikt, Egor Kostylev, Fabio Mogavero, and Efthymia Tsamoura. Reformulating queries: theory and practice. In *IJCAI*, 2017.
[4] Michael Benedikt, Julien Leblay, Balder ten Cate, and Efthymia Tsamours. *Generating Proof from Plans: the interpolation-based approach to Query Reformulation*. Morgan Claypool, 2016.
[5] Michael Benedikt, Julien Leblay, and Efi Tsamoura. PDQ: Proof-driven query answering over web-based data. In *VLDB*, 2014.
[6] Michael Benedikt, Julien Leblay, and Efthymia Tsamoura. Querying with access patterns and integrity constraints. *PVLDB*, 8(6):690–701, 2015.
[7] Michael Benedikt, Rodrigo Lopez-Serrano, and Efthymia Tsamoura. Biological web services: Integration, optimization, and reasoning. In *Advances in Bioinformatics and Artificial Intelligence: Bridging the Gap*, 2016.
[8] Michael Benedikt, Balder ten Cate, and Efi Tsamoura. Generating low-cost plans from proofs. In *PODS*, 2014.
[9] Michael Benedikt, Balder ten Cate, and Efi Tsamoura. Generating plans from proofs. In *TODS*, 2016.
[10] E. W. Beth. On Padoa's method in the theory of definitions. *Indagationes Mathematicae*, 15:330 – 339, 1953.
[11] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.
[12] Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3):200–226, 2007.
[13] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *PODS*, 2008.
[14] Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.
[15] Tomasz Gogacz and Jerzy Marcinkowski. The hunt for a red spider: Conjunctive query determinacy is undecidable. In *LICS*, 2015.
[16] Ioana Ileana, Bogdan Cautis, Alin Deutsch, and Yannis Katsis. Complete yet practical search for minimal query reformulations under constraints. In *SIGMOD*, 2014.
[17] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *PODS*, 1995.
[18] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *AAAI*, 1996.
[19] Chen Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.
[20] Chen Li and Edward Chang. Answering queries with useful bindings. *TODS*, 26(3):313–343, 2001.
[21] Roger C. Lyndon. An interpolation theorem in the predicate calculus. *Pacific Journal of Mathemantics*, 9:129–142, 1959.
[22] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *TODS*, 4(4):455–469, 1979.
[23] Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.
[24] Alan Nash and Bertram Ludäscher. Processing union of conjunctive queries with negation under limited access patterns. In *EDBT*, 2004.
[25] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.
[26] Lucian Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, U. Penn., 2000.
[27] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 10(2-3):182–198, 2001.
[28] Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3), 2008.
[29] Luc Segoufin and Victor Vianu. Views and queries: determinacy and rewriting. In *PODS*, 2005.
[30] David Toman and Grant Weddell. *Fundamentals of Physical Design and Query Compilation*. Morgan Claypool, 2011.
[31] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, V2*. Comp. Sci. Press, 1989.

Data Lifecycle Challenges in Production Machine Learning: A Survey *

Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang †, Martin Zinkevich

Google Research KAIST†

{npolyzotis, sudipr, martinz}@google.com swhang@kaist.ac.kr†

ABSTRACT

Machine learning has become an essential tool for gleaning knowledge from data and tackling a diverse set of computationally hard tasks. However, the accuracy of a machine learned model is deeply tied to the data that it is trained on. Designing and building robust processes and tools that make it easier to analyze, validate, and transform data that is fed into large-scale machine learning systems poses data management challenges.

Drawn from our experience in developing data-centric infrastructure for a production machine learning platform at Google, we summarize some of the interesting research challenges that we encountered, and survey some of the relevant literature from the data management and machine learning communities. Specifically, we explore challenges in three main areas of focus – data understanding, data validation and cleaning, and data preparation. In each of these areas, we try to explore how different constraints are imposed on the solutions depending on where in the lifecycle of a model the problems are encountered and *who* encounters them.

1. INTRODUCTION

Machine learning (ML) has become essential in modern computing. More and more organizations are adopting ML to glean knowledge from data and tackle a diverse set of computationally hard tasks, ranging from machine perception and text understanding to health care and genomics. As a striking example, deep learning techniques can be used to detect diabetic eye diseases with an accuracy on-par with ophthalmologists [1].

However, developing reliable, robust, and understandable ML models requires much more than a good training algorithm. Specifically, it is necessary to build the model using high-quality training data. Moreover, this training data needs to be translated into a set of features that can expose the underlying signal to the training algorithm. And finally, the data fed to the model at serving time must be similar in distribution (and in features) to the training data, otherwise the model's accuracy will decrease. Ensuring that each of these steps is done in a consistent manner becomes even more challenging in a setting where new training data arrives continuously and accordingly triggers the training and deployment of updated models.

To further illustrate the previous points, we consider a scenario where a software error in a data source causes a feature in the training data to get pinned to an error value (e.g., -1). Training on such corrupted data will typically lead to reduced model accuracy that may only be noticed after a few days. The predictions obtained using the poor model will persist in the logged serving data (new data on which the model runs on). Typically this logged data is fed back as training data for the next training cycle. This can therefore cause the data error to percolate through the system and taint downstream data, which can make recovery painful. Depending on the impact of the error on the model accuracy, it can at best cause a small reduced model accuracy and at worst cause a hard to recover service outage. This scenario illustrates the importance of catching errors early and reasoning about their propagation within the data flow of an ML pipeline.

In this article, driven from our experience in building data management infrastructure for a large-scale ML platform [11], we identify several core challenges in the management of ML data that are relevant to [11] and other ML platforms [17]. We organize these challenges around the following broad themes: data understanding, data validation and cleaning, and data preparation. We draw connections to existing work in the data management literature and outline open problems that remain to be solved.

*This article extends a tutorial the authors delivered at the SIGMOD conference in 2017.

†Corresponding author, work done at Google and KAIST

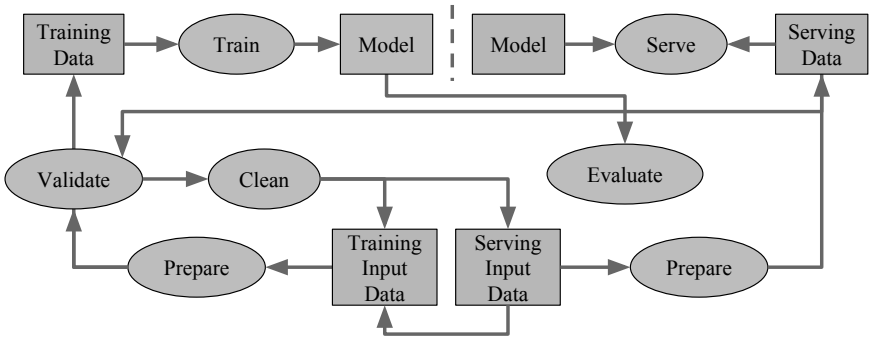


Figure 1: An overview of an end-to-end machine learning pipeline with a data point of view.

The rest of the paper is structured as follows:

- We provide an overview of large-scale ML pipelines through the lens of data management (Section 2).
- We focus on the following data management themes and study existing work: data understanding (Section 3), data validation and cleaning (Section 4), and data preparation (Section 5). At the end of each section, we identify open research challenges.
- We summarize lessons learned through our experience in building data management infrastructure for large-scale ML (Section 6).

The article aims to inform both database researchers and practitioners about the class of problems that exist in the intersection of production ML pipelines and data management, and to motivate further research in this area. We believe that the database community is well positioned to tackle these problems in the context of ML.

2. OVERVIEW OF PRODUCTION ML

The data management community has explored several interesting problems around the optimization of ML pipelines as data flows, and this line of work has resulted in the development of novel system architectures such as Velox [24], Weld [54], and SystemML [15]. In comparison this survey takes a *data centric point of view* and focuses on the challenges that arise in the management of ML data, which are largely separate from the efficiency issues of large-scale data flows.

More recently model understanding has become a critical issue especially when using deep learning or representative learning on semi-structured or unstructured data. The challenges range from understanding the state at different layers of deep architectures, interpreting results [28], to finding minimal architectures without reducing model accuracy. Model understanding deserves a survey on its own, and this survey complements by focusing more on training and serving data.

In this section, we introduce two dimensions that help us characterize data management challenges. The first dimension derives from the different classes of users of an ML pipeline. The second dimension stems from the data's lifecycle through an ML pipeline and the corresponding activities performed by the users. The following subsections discuss these dimensions in more detail.

2.1 Users Interacting with ML Platforms

An often overlooked aspect of large-scale ML at bigger companies is that multiple people play different roles [41] in the development and maintenance of an ML pipeline. The best person for coming up with new features is unlikely to be the best person to develop the ML architecture or to handle emergencies with the production system: moreover, people will approach the pipeline with different priorities. Borrowing from marketing and UX research, we identify three personas [37] representative of groups that would use an ML pipeline based on our experience:

- ML Expert: has a broad knowledge of ML, knows how to create models and how to use statistics, and can advise multiple pipelines.
- Software Engineer: understands the problem domain and has the most engineering expertise for a specific product.
- Site Reliability Engineer: maintains the health of many ML pipelines simultaneously, but cannot afford to know the application details.

As an example of how these personas play different roles, suppose that the pipeline is experiencing new errors due to an out-of-range feature value (e.g., the price of an item is higher than expected). The ML expert could fix the quantization of the price for model training. The software engineer could implement the quantization and run backfilling (explained in Section 2.2). The site reliability engineer, on the other hand, may want to rollback the pipeline to a working state first.

Finally, the three personas play different roles in an ML pipeline's lifecycle. During the experiments phase, the ML expert and software engineer are mostly involved. During the launch, the site reliability engineer becomes involved. The subsequent refinement of the pipeline is done by the software engineer while the maintenance of the overall pipeline is done by the site reliability engineer. A key takeaway is that many people with radically different backgrounds will have to handle a variety of tasks to keep the pipeline running smoothly.

2.2 Data Lifecycle in an ML Pipeline

The data lifecycle of an ML pipeline starts with generating *Training data*. Specifically, we distinguish the raw *input data* that is fetched from a variety of sources including databases, key-value stores, and logs, to name a few. Depending on the type of the problem at hand and available data, this data can be structured, semi-structured, or unstructured, and may correspond to different degrees of curation. In many cases, a few invariants can be asserted over the data [62].

As a running example, suppose a team is developing an app store. The initial raw input data is illustrated on the left side of Figure 2, and the goal is to predict app purchases based on app store user and product features. (Note that the "app store users" are not the same as "users" of the ML pipeline.) While the example is intentionally simplistic, the input data can be large and generated by joining heterogeneous data sources with different data qualities and trust issues.

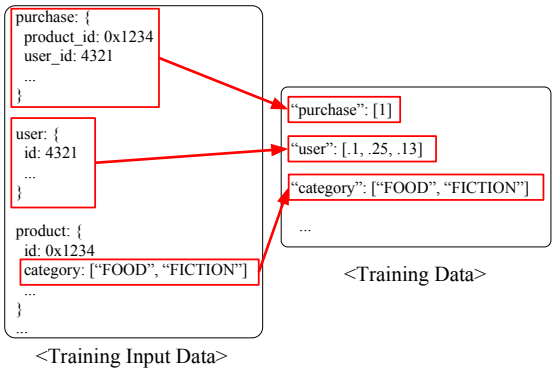


Figure 2: Input to Training Data for the App Store ML pipeline. The same preparation must also be done for serving data as shown in Figure 1.

Prepare. The input data is transformed to the training data through the *Prepare* module. Figure 2 shows how the raw input data is converted to a format of features and values, which can be used for training by the *Train* module. For example, the user information has been mapped to a vector of three values, which is called an embedding. The key questions to ask for preparation are: what features can be generated from data, what are the properties of the feature values, and what are the best practices to transcode values.

Train and Evaluate. Once the training data is ready, it is fed into the *Train* module, which can be frameworks including TensorFlow [5], Keras [3], and Apache MXNet [4]. The trained model can be evaluated by an *Evaluate* module, which checks if the model has an acceptable accuracy, whether the data should be encoded differently, and whether there should be more data or features. Note that an ML pipeline may train an ensemble of models using either the same or different input data where the predictions are intersected or unioned to increase accuracy.

Validate. The *Validate* module is necessary to make sure the training data does not contain errors that may propagate down to the model training. To illustrate why validating data is important, suppose that an engineer is refactoring a backend that generates a feature for the app store ML pipeline, but introduces a bug that results in the generation of wrong values. Note that in this case there are no newly introduced features or data, and the training and serving logic remain the same. Once the erroneous code rolls out to production, it causes the feature to acquire erroneous values. The bad feature values then cause the model training accuracy to decrease and result in significant production issues during model serving where the model is executed for the app store users. Overall, data validation is a key element of production ML infrastructure: by detecting an issue during training time, we can avoid the rollout of a broken model and thus prevent a significant negative impact on the app store users and revenue.

Validating data is a complex problem that needs to be solved for various parts of the pipeline. In the *Prepare* module, the key questions to ask are which data properties affect significantly the model accuracy and whether there are any dependencies to other data and infrastructure. Validation is also required between the *Training Data* and *Serving Data* (depicted as the arrow from *Serving Data* to *Validate* in Figure 1). If there are deviations between the two types of data, then the model trained on the training data will not perform consistently during serving. Hence, the questions to ask are what are possible deviations between the two types of data and when they can be problematic.

Any detected data errors must be forwarded to the user as alerts. An important challenge is to formulate alerts so they are understandable and actionable. As an example, consider the detection of a missing feature Country from the app store user data and the following alternatives to formulate the alert: (a) feature Country is missing, (b) feature Country is missing from 18% of the input examples, or (c) feature Country is missing from the examples that correspond to "gender=female". Clearly, some of these formulations are more actionable and allow the user to understand better both the scope of the error and its potential effect on model accuracy. Another issue is how sensitive the alerts should be. If there are too many false positives, the user may end up ignoring alerts altogether. On the other hand, being too strict on alerts will result in failing to detect critical errors.

Clean. The user may also decide to fix the data based on the alerts, after understanding whether cleaning the data will improve the model, which part of the data is to be fixed, and how should the fix be reflected to all the input data until now (this operation is known as backfilling).

Serve. After a model is trained and deployed, the *Serve* module is responsible for receiving the *serving input data* (in our example, app store user impressions and clicks) and preparing it as *serving data* that can be processed through the model. The serving input data is typically generated a single example at a time, and has stringent latency constraints. The serving input data needs the same preparation as was applied to the raw training time before being sent to the model.

It is often the case that serving data is logged and channelled back as training data for the next training epoch through some bulk data processing stages, thereby completing the data lifecycle.

In summary, there are various data management challenges that arise at different stages of the data lifecycle, which we broadly classify as follows:

- Data Understanding: analyzing and knowing what to expect from the data.
- Data Validation and Cleaning: identifying and fixing any errors in the data.
- Data Preparation: engineering features and gathering examples for training.

3. DATA UNDERSTANDING

The first step of ML is to understand your data. There are largely two parts in an ML pipeline for data understanding. First, sanity checks are important when training a model for the first time. Next, more advanced analysis and diagnosis are needed during the launch and iterate cycles where a model is iteratively improved with new training data.

3.1 Sanity Checks

When the user performs sanity checks, the challenge is to see if the data has the expected "shape" before training the first model. The following are some examples of sanity checks:

- A continuous feature's minimum, maximum, most common values, and histogram are reasonable (e.g., latitude values must be within the range [-90, 90] or $[-\frac{\pi}{2}, \frac{\pi}{2}]$, and not all values are in one bucket).
- The distribution of a categorical value is as expected (e.g., it has the expected domain, and the more common values are what you would expect).
- A feature is present in enough examples (e.g., the country code must be in $\geq$70% of the examples).
- A feature has the right number of values (e.g., there cannot be more than one age of a person).
- Labels from external services may have trust issues and must be verified with known labels.

The key ML challenge here is how to set such expectations of the data. For example, how do we know a distribution is "right"? If we know exactly what we need, then one can use any SQL tool to perform sanity checks. However, the requirements are often unclear because there may be no ownership of the feature. In this case, visualization tools can help us understand the data shape by discovering surprising properties of data and thus develop better sanity checks.

SeeDB [70] recommends data-driven visualizations using deviation-based metrics (e.g., Earth Mover's distance, Euclidean distance, Kullback-Leibler divergence, and Jenson-Shannon distance). The recommendations can provide insights to users on what to expect of the training data and subsequent ones. For example, suppose there are two histograms that show Desktop versus Mobile usage between two groups of people. If the two groups are female versus male, the two histograms may not differ much. However, if the groups are users in emerging versus mature markets, there may be a relatively higher mobile usage in emerging markets. This difference in usage makes the latter histogram more interesting and thus more likely to be recommended. ZenVisage [66] is a follow-up work on interactive visual analytics using the ZQL query language.

Another recent line of work is to control false discovery rates for recommending visualizations. Con-

tinuing our example above, as more visual recommendations are made, there is bound to be more meaningless ones as well. The QUDE system [14, 75] takes a statistical approach and provides an interactive data exploration framework that uses multiple hypothesis testing to control the false positives. Traditional methods for controlling family-wise error rates (e.g., Bonferroni correction) or false-discovery rates (e.g., Benjamini-Hochberg procedure) assume "static" hypotheses and do not work for interactive data exploration. QUDE proposes α-investing to control the marginal false discovery rate, which is the expected value of the false discovery rate. Intuitively, recommending good visualizations will be rewarded with more budget to explore while bad recommendations will result in losing it.

3.2 Analyses for Launch and Iterate

The next part of data understanding is to do more analyses during the launch and iterate cycles.

3.2.1 Feature-based Analysis

There are major ML challenges that involve feature analysis. One is analyzing features in conjunction with a trained model where the goal is to find interesting training data slices (based on features) that lead to high/low model accuracy. For example, an app recommendation model may perform poorly for people in certain countries. Another challenge is detecting training-serving skew, which was briefly mentioned in Section 1. For instance, if the model was trained on data that had an even gender ratio, but the actual serving logs have a completely different ratio for people in the age range [20, 40], then the model may be biased due to the skew. As another example, there may be unseen features that appear on serving data, but not in training data. Skew can be fixed by debugging data generation, which is usually the culprit, or possibly making the model training more robust to skew.

Data cube analysis can be applied to analyze slices of data, which are defined with features or feature crosses. For example, MLCube [39] is a tool for visually exploring ML results that enable users to define slices using feature conditions to compute aggregate statistics and evaluation metrics over the slices. The tool can be used to help understand and debug a model or compare two models. Another interesting work is prediction cubes [21], which summarize models trained on individual cubes.

While such manual exploration is useful, an interesting research question is how to automatically prioritize user attention and identify what are the "important" slices. While we are not aware of any recent data cube research that directly addresses this problem, intelligent roll-ups in multi-dimensional OLAP data [60] are relevant and have been proposed to automatically generalize from a specific problem case in detailed data and return the broadest context in which the problem occurs. Similarly, smart drill-downs [38] discover and summarize interesting slices of the entire data. The roll-ups or drill-downs can be used to find problematic slices in training data that positively (or negatively) affect model metrics (e.g., log loss, AUC, and calibration).

3.2.2 Data Lifecycle Analysis

Another important analysis is to track the lifecycle of data. A common analysis is to identify dependencies of features. For example, a label feature must not "leak" into any other feature where some of its information is duplicated or encoded in the other feature, and the model trained on that information makes unrealistically-accurate predictions, but generalizes poorly. Another useful analysis is to identify sources of data errors. For example, a subset of the training data may have been dropped because a data source was unavailable. The tools to address these analyses largely fall into two categories: coarse-grained and fine-grained tracking.

The advantage of coarse-grained tracking is that it is general and not tied to a particular system. Goods [35] gathers metadata from tens of billions of datasets (including provenance) within Google and implements services on top of this metadata. A key design choice is to gather this data in a post-hoc fashion where dataset owners do not have to do any registration, and the metadata is crawled afterwards in a non-intrusive manner. As a result, while Goods can track which dataset was generated from which process, it cannot extend the tracking to individual features.

Fine-grained tracking, on the other hand, can analyze individual features, but tends to be tightly coupled with the underlying system. ProvDB [49] provides a unified provenance and metadata management system to support lifecycles of complex collaborative data science workflows. The metadata consists of artifacts, which include version lineages of data, scripts, results, data provenance among artifacts, and workflow metadata on derivations and dependencies among artifact snapshots. Other relevant systems include ModelDB [69], ModelHub [50], and Amazon's ML experiments system [61], which provide lifecycle management for various models, and Ground [36], which has a goal similar to that of ProvDB, but with a simple, flexible metamodel that is model agnostic.

3.3 Open Challenges

There are open questions for ML analysis that are not covered by the previous techniques. Recently, determining if a trained model is "fair" [59] has become a critical issue. For example, we would like to know if a model is prejudiced against certain classes of data. Since a model is only as good as its training data, we need to understand if the data reflects reality. Identifying new kinds of "spam" [34] is an open challenge as well. For example, are users abusing the system in an adversarial way? Here, we need to apply adversarial testing on the training data. While using general SQL-based systems [48, 6] is an "escape hatch" for analysis, we may need more specialized tools to address the above issues.

4. DATA VALIDATION AND CLEANING

Since ML largely depends on its data, it requires data validation to perform well. Models cannot answer questions they are not asked. For example, suppose a model uses the feature Country and understands when its value is "US". However, if in the next batch of training data the value becomes "us", then without any validation or preprocessing, the model will simply think that there is a new country. As another example, a feature may suddenly change its unit (e.g., age changes from days to hours) or even disappear. Unfortunately, model training is resilient to such errors, and instead of crashing, the model accuracy may simply decrease. Hence, data must be validated early on to avoid errors from propagating to model training.

How do we deal with these problems? If a feature value is not consistent, we could insert automatic corrections (e.g., capitalize all countries). If a feature appears for the first time, we can create a new field. If a feature disappears, we can find where it disappeared using provenance or root cause analysis [53, 72, 9]. While some fixes can be done automatically, in many cases, we need to notify users to solve the problems by providing "playbooks," which are manuals that contain concrete actions to take for addressing each alert. Although many data cleaning [68, 40, 45] techniques are relevant, in production ML it is equally important to design actionable alerts with humans in mind.

4.1 Alert Tradeoffs

When alerting users to validate and fix data errors, there is a tradeoff to make between recall and precision. It is not rare for some ML data issue to cause a minor emergency for a particular product. A common response is to overcompensate, by setting alerts for every conceivable issue with the data. Then, these alerts fire every day, annoying users and making them insensitive to real issues. As a result, all the alerts are ignored, and the vicious cycle starts again. Hence, it is important to balance recall (i.e., what fraction of problems we catch) and precision (what fraction of alerts lead to good catches).

An alert is considered a good catch if it is actionable and eventually leads to a fix. For example, alerting that a feature is missing is clearly actionable. However, alerting that there was a distributional shift in a feature's value may be less actionable. Imagine an ML expert saying that the age should have a Kolmogorov distance of less than 0.1 from the previous day and then leaves and works on a different system. Later on, another engineer may be alerted that the age has a Kolmogorov distance of 0.11. While the alert may indicate a real problem, the engineer may not know how to resolve it. Hence, the question is not whether something is wrong if an alert fires, but whether it gets fixed.

In some cases, a data fix may encompass fixing a constraint. For example, suppose a Country feature is known to contain four values, but from some point a new value "SS" is introduced in the training data. While this value may indeed be incorrect, "SS" could also be a valid country (say South Sudan), which means the constraint should now include five countries. Existing work [22, 33] provides opportunities to fix both data and constraints.

If there are many alerts, finding alerts that are related and combining them becomes useful. In the literature, cost-based models [16, 42] and conflict hypergraphs [23] are proposed.

Not all anomalies are equally important, and the ones that result in worse model accuracy in production must be alerted first. In large-scale ML, features with different qualities co-exist. Often, an "alpha" or completely new and untested features will co-exist with more established production features that the system relies on. An untested feature is evaluated in an experimental model, which may become the next production model. At some point, a feature may also be deprecated. Hence, fixing features that are used in production is more useful than fixing features for experimental models. An interesting open question is whether the improvements of correcting a feature can be estimated without having to make the correction itself.

4.2 Alert Categories

General alerts are hard to design and depend on the training data. For example, predicting car accidents, house prices, social connections, or clicks on a web page will have very different data, and it

is hard to predict the expected data shape for all these applications. To handle a variety of domains with minimal effort, there needs to be some commonalities among them.

One common setting is where data arrives continuously, say in web applications. As new data arrives, old data is thrown away, and newer data is given more priority. The data validation can be done by comparing data with its previous versions, and alerts can be raised based on accumulated evidence to date [71]. In case there is a concept drift, the expected data shape can be updated based on the user's judgement.

In this setting, basic alerts are motivated by engineering problems. For example, missing fields can be detected by checking if a field that was present is now absent. RPC timeouts can be detected by checking if the most common value is not more common than before. Format changes can be detected by checking if the domain of values has increased.

It is also worth mentioning alerts that are based on statistics including homogeneity tests, analysis of variance (e.g., ANOVA [30, 31]), time series analysis, and change detection. For example the chi-squared test can be used to check homogeneity [55] by rejecting the null hypothesis for the distributions being the same. One problem with a chi-squared test is that statistically significant changes may be common on all fields if the data is large enough. Other metrics that can be used include the L_1 and L_∞ metrics or Earth Mover's distance [32, 70]. Some metrics including the number of examples, the number of positive labels, or the total number of clicks may fluctuate, but are nonetheless very important. Time series analysis [10, 27, 18] can help track these statistics.

4.3 Open Challenges

Selecting alerts that lead to the most impact in production is an open question. Ideally, we can perform impact analysis that will estimate how the system would improve if an error were fixed. Automatically generating fixes and playbooks is also an open-ended challenge. There is an interesting connection between the notion of alerts and fixes to existing work on automatic database repairs. Similarly, the notion of minimizing the number of alerts to users is analogous to active learning, which tries to minimize the number of labelings.

5. DATA PREPARATION

One of the largely "black art" aspects of ML is data preparation. Similar to the other data management challenges, specific facets of the data preparation problem arise in different forms at different points in the ML lifecycle. For instance, during the initial development of a model, data preparation boils down to engineering a set of features that are most predictive of the task label. Once the models are more mature, the focus may shift to resource optimization and latency reduction by selecting a subset of all the available features while still retaining the same accuracy. This is typically referred to as the *feature selection* problem. Finally, often the original data that was available for the task may simply be incomplete or partially complete. A third aspect of data preparation is enriching the training data by importing information from other data sources.

5.1 Feature Engineering and Selection

Feature engineering is a well-studied problem [7, 44, 43, 74, 58, 46, 8, 67] with a suite of techniques that are largely designed based on experience of ML experts. Consider the following specific ML task. Starting with the census data on housing, we would like to predict the median housing prices at the granularity of city blocks. For this task, reasonable features include location of blocks (possibly specified using latitude and longitude coordinates), number of households per block, crime rate, and so on. While some of these features may directly be available in the census data, others may require some queries over the data to extract. Furthermore, the *goodness* of a feature is typically based on the predictive power of the feature. While the predictive power is hard to estimate upfront, a good proxy is to understand the correlation of the feature with the label. Analyses techniques discussed in Section 3 that easily present these correlations to experts can be invaluable here.

Even once the raw features are designed, often a suite of transformations are applied before they are fed into the ML pipeline. Some of the typical transformations applied include normalization, bucketization, winsorizing, one-hot encoding, feature crosses, and using a pre-trained model or embeddings (mappings from values such as words to real numbers) to extract features [51]. For example, the crime rate of a city block could be transformed into a one-hot encoding using three categories: low, medium, and high. The exact feature transform to perform depends on both the data as well as the ML training algorithm. Some algorithms that natively perform transformation include Lasso (regularize unimportant features to zero) and on-the-fly scaling and shifting (avoid turning sparse data into dense data).

An interesting research direction is to learn feature engineering itself. Feeding training data directly to a deep neural network and letting it figure out the features is referred to as "representation learning" in the ML community. Some promising techniques include autoencoders and restricted Boltzmann Machines [12]. However, learning both the representations and the objective may require significant resources and data, so manual feature engineering is still used in most cases. A middle ground between completely automated feature engineering and manual feature engineering is data-driven feature engineering where based on certain data characteristics we can automatically infer the best set of transformations to apply. While this is relatively simple to do in many cases, determining the best embedding that should be used on a feature by searching over an available set of pre-trained embeddings is still an open research problem.

As models become more mature, developers often experiment with addition and removal of new features. This is one instance of a problem that highlights the different views that users with different roles have for the same problem. For example, an ML expert could be choosing features that improve the model accuracy the most. However, the software engineer may have to also worry about how to actually add the feature into the existing pipeline. The check list includes making sure the feature is available at serving time, whether one is allowed to even use the feature, and what the return of investment is for the feature. The site reliability engineer may have to worry about introducing new dependencies and making sure the pipeline is robust. Another concern is whether the feature will affect the model size and prediction latency.

5.2 Data Enrichment

Data enrichment [29, 56, 64] refers to the augmentation of the training and serving data with information from *external data sources* in order to improve the accuracy of the generated model. A common form of enrichment is to join in a new data source in order to augment the existing features with new signals. Another form is using the same signals with different transformations, e.g., using a new embedding for text data.

A first step for data enrichment is cataloging and contextualizing all the available data. Different systems have been designed that solve this problem within enterprises as well as over the web. For example, systems including Goods [35], Ground [36], and Datahub [13] can be used to explore datasets siloed within product areas of organizations. On the web, tools including Webtables [19], Kaggle [2], and Data Civilizer [20] can be used to search scientific datasets published independently by organizations.

The next step is to extract knowledge and acquire labels, which can be challenging when labeling is expensive and/or heterogeneous sources can have different label qualities and labeling costs. Crowdsourcing frameworks like Alfred [25] help moderately skilled crowd workers in extracting knowledge from a corpus of unstructured documents. DeepDive [73] uses incomplete knowledge bases and rules for distant supervision to minimize expensive human annotations. In active learning [26, 63, 52], the learning procedure decides how best to enrich the data iteratively. Transfer learning is a way to leverage previously acquired knowledge from one domain to improve the model accuracy of a different domain. Weak supervision is used in Snorkel [57, 56] where hand labeling is avoided altogether, and workers can programmatically generate lower-quality labels, which are then denoised with generative models. Finally, label hierarchies can be used to predict unseen labels.

While data enrichment often leads to model accuracy improvements, this is not always the case. Sheng et al. [65] shows how improving the quality of the already available labels can better improve model accuracy than collecting more examples. Hence, there may be a tradeoff between obtaining more data and improving its data. Similary, a recent paper [47] has looked into understanding the impact of adding new features by joining with other data sources for a specific class of algorithms. It would be interesting to consider extensions to other cases (e.g., training with black-box learning algorithms that are hard to approximate, or using different transformations on existing signals).

5.3 Open Challenges

Given input features and an ML training algorithm, automatically generating feature transforms that result in the highest model accuracy is an open question. From our experience, this step is a pain point for users who do not necessarily understand the nuances of transforms.

6. LESSONS LEARNED

In building a production ML platform for Google [11], we encountered a host of the challenges that we have presented in this survey. We summarize some of the over-arching lessons that we learnt on the way.

- *Interesting data management challenges beyond optimizing data flow.* The data management community has focused more on optimizing data flow

for large scale data processing (specifically for ML). However, as we discussed in this article, there are data management problems beyond optimizing data flow. Complementing the traditional data flow point of view, we have provided a data point of view for ML pipelines and identified challenges in understanding, validating, and preparing data. As shown in the previous sections, many prior techniques from the data management literature are relevant to building robust large-scale ML systems. The data management and ML communities have a lot to learn from each other through closer collaboration.

- *Make realistic assumptions when developing solutions.* In developing research solutions, we must be careful about the assumptions that we make. For instance, it is unreasonable to assume that data lives in a single source (e.g., a DBMS). Instead, most enterprise data often resides in multiple storage systems (e.g., Spanner, BigTable, Dremel, and CNS, to name a few) that have different characteristics. Typically, there needs to be an ingestion step that converts this data to become compatible with the trainer. Similarly, it is important to stay abreast of the state-of-art developments in the ML community and ensure that the data management solutions complete them.
- *Be aware of the diverse needs of different users.* Many of the key design decisions in our infrastructure were based on diverse needs of different set of users that interact with such a system. In addition to the personas covered in this paper, ML systems may have a wide spectrum of end users as well, starting from ML novices who have yet to train their first models up to experts with extensive modeling experience. Building a large-scale ML system must be flexible enough to accommodate all these users as much as possible.
- *Ensure that your solution integrates smoothly into the development workflow.* The launch and iterate cycle time for ML pipelines is small, and users will not use tools unless they are necessary for their critical development workflows. To ensure the adoption of data management tools, it is thus critical to integrate them into workflows smoothly and make the benefits of using them obvious.

7. CONCLUSION

Data management in large-scale ML systems will only get more important as the amount of data continues to increase rapidly. In this survey, we have described large-scale ML pipelines in a data point of view. We then focused on three data management challenges – understanding, validation and cleaning, and preparation – and surveyed relevant techniques from the data management literature. Finally, we summarized lessons learned from building data management infrastructure for a large-scale ML platform. We believe data management research in ML systems has plenty of open challenges that can be solved with close collaboration between the data management and ML communities.

8. REFERENCES

[1] Deep learning for detection of diabetic eye disease. https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html.

[2] Kaggle. https://www.kaggle.com/.

[3] Keras. https://keras.io/.

[4] Mxnet. https://mxnet.incubator.apache.org/.

[5] Tensorflow. https://www.tensorflow.org/.

[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eurosys*, pages 29–42, 2013.

[7] M. R. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[8] M. R. Anderson and M. J. Cafarella. Input selection for fast feature engineering. In *ICDE*, pages 577–588, 2016.

[9] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. Macrobase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[10] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application.* Prentice-Hall, Inc., 1993.

[11] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*, pages 1387–1395, 2017.

[12] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828, 2013.

[13] A. P. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative

data science & dataset version management at scale. *CoRR*, abs/1409.0798, 2014.
[14] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*, 2017.
[15] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, and S. Tatikonda. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425–1436, 2016.
[16] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.
[17] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. *PVLDB*, 10(12):1694–1705, 2017.
[18] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. Inferring causal impact using bayesian structural time-series models. *Annals of Applied Statistics*, 9:247–274, 2015.
[19] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.
[20] R. Castro Fernandez, D. Deng, E. Mansour, A. A. Qahtan, W. Tao, Z. Abedjan, A. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. A demo of the data civilizer system. In *SIGMOD*, pages 1639–1642, 2017.
[21] B.-C. Chen, L. Chen, Y. Lin, and R. Ramakrishnan. Prediction cubes. In *PVLDB*, pages 982–993, 2005.
[22] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.
[23] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[24] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR*, 2015.
[25] V. Crescenzi, P. Merialdo, and D. Qiu. Crowdsourcing large scale wrapper inference. 33:1–28, 2014.
[26] S. Dasgupta and J. Langford. Tutorial summary: Active learning. In *ICML*, page 18, 2009.
[27] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.
[28] F. Doshi-Velez and B. Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017.
[29] R. C. Fernandez, Z. Abedjan, S. Madden, and M. Stonebraker. Towards large-scale data discovery: Position paper. In *ExploreDB*, pages 3–5, 2016.
[30] R. A. Fisher. On the probable error of a coefficient of correlation deduced from a small sample. *Metron*, 1:3–32, 1921.
[31] R. A. Fisher. *Statistical Methods for Research Workers*, pages 66–70. Springer New York, 1992.
[32] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review*, 70(3):419–435, 2002.
[33] L. Golab, I. F. Ilyas, G. Beskales, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.
[34] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.
[35] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD*, pages 795–806, 2016.
[36] J. M. Hellerstein, V. Sreekanti, J. E. Gonzales, Sudhansku, Arora, A. Bhattacharyya, S. Das, A. Dey, M. Donsky, G. Fierro, S. Nag, K. Ramachandran, C. She, E. Sun, C. Steinbach, and V. Subramanian. Establishing common ground with data context. In *CIDR*, 2017.
[37] A. Jenkinson. Beyond segmentation. *Journal of Targeting, Measurement and Analysis for Marketing*, (1):60–72, 1994.
[38] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, pages 906–917, 2016.
[39] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *HILDA*, pages 1:1–1:6, 2016.
[40] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden,

M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[41] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *TSE*, PP(99):1–1, 2017.

[42] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[43] P. Konda, A. Kumar, C. Ré, and V. Sashikanth. Feature selection in enterprise analytics: A demonstration using an r-based data analytics system. *PVLDB*, 6(12):1306–1309, 2013.

[44] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[45] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[46] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Rec.*, 44(4):17–22, 2016.

[47] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD*, pages 19–34, 2016.

[48] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1-2):330–339, 2010.

[49] H. Miao, A. Chavan, and A. Deshpande. Provdb: A system for lifecycle management of collaborative analysis workflows. *CoRR*, abs/1610.04963, 2016.

[50] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. In *ICDE*, pages 571–582, 2017.

[51] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[52] F. Olsson. A literature survey of active machine learning in the context of natural language processing. volume T2009 of *SICS Technical Report*. Swedish Institute of Computer Science, 2009.

[53] C. Olston and B. Reed. Inspector gadget: A framework for custom monitoring and debugging of distributed dataflows. In *SIGMOD*, pages 1221–1224, 2011.

[54] S. Palkar, J. J. Thomas, A. Shanbhag, M. Schwarzkopt, S. P. Amarasinghe, and M. Zaharia. A common runtime for high performance data analysis. In *CIDR*, 2017.

[55] K. Pearson. *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to have Arisen from Random Sampling*, pages 11–28. Springer New York, 1992.

[56] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

[57] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS*, pages 3567–3575, 2016.

[58] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base construction. *IEEE Data Eng. Bull.*, 37(3):26–40, 2014.

[59] A. Romei and S. Ruggieri. A multidisciplinary survey on discrimination analysis. *Knowledge Eng. Review*, 29(5):582–638, 2014.

[60] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, 2001.

[61] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert. Automatically tracking metadata and provenance of machine learning experiments. In *Workshop on ML Systems at NIPS 2017*, 2017.

[62] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *NIPS*, pages 2503–2511, 2015.

[63] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2012.

[64] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *PVLDB*, 11(3):366–379, 2017.

[65] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *SIGKDD*, pages 614–622, 2008.

[66] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data

exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

[67] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *ICDE*, pages 535–546, 2017.

[68] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[69] M. Vartak. MODELDB: A system for machine learning model management. In *CIDR*, 2017.

[70] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[71] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

[72] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245, 2015.

[73] C. Zhang. *DeepDive: A Data Management System for Automatic Knowledge Base Construction*. PhD thesis, 2015.

[74] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *ACM TODS*, 41(1):2:1–2:32, 2016.

[75] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *SIGMOD*, pages 527–540, 2017.

Stream Processing Languages in the Big Data Era

Martin Hirzel
IBM Research, USA
hirzel@us.ibm.com

Guillaume Baudart
IBM Research, USA
Guillaume.Baudart@ibm.com

Angela Bonifati
Lyon 1 University, France
angela.bonifati@univ-lyon1.fr

Emanuele Della Valle
Politecnico di Milano, Italy
emanuele.dellavalle@polimi.it

Sherif Sakr
University of Tartu, Estonia
sherif.sakr@ut.ee

Akrivi Vlachou
University of Piraeus, Greece
avlachou@aueb.gr

ABSTRACT

This paper is a survey of recent stream processing languages, which are programming languages for writing applications that analyze data streams. Data streams, or continuous data flows, have been around for decades. But with the advent of the big-data era, the size of data streams has increased dramatically. Analyzing big data streams yields immense advantages across all sectors of our society. To analyze streams, one needs to write a stream processing application. This paper showcases several languages designed for this purpose, articulates underlying principles, and outlines open challenges.

1. INTRODUCTION

We have entered the big-data era: the world is awash with data, and more data is being produced every second of every day. Data analytics solutions must contend with data being *big* both in the static-data sense of an ocean of many bytes and in the streaming sense of a firehose of many bytes-per-second. In fact, driven by the realization that static data is merely a snapshot of parts of a data stream, the data technology industry is focusing increasingly on data-in-motion. Analyzing the stream instead of the ocean yields more timely insights and saves storage resources [6].

Stream processing languages facilitate the development of stream processing applications. Streaming languages simplify common coding tasks and make code more readable and maintainable, and their compilers catch programming mistakes and apply optimizing code transformations. The landscape of streaming languages is diverse and lacks broadly accepted standards. Stephens [79] and Johnston et al. [56] published surveys on stream processing languages in 1997 and 2004. Much has happened since then, from database-inspired streaming languages to the rise of big data and beyond. Our survey continues where prior surveys left off, focusing on streaming languages in the big-data era.

A *stream* is a sequence of data items, and the length of a stream is conceptually infinite, in the sense that waiting for it to end is ill-defined [70]. A streaming application is a computer program that consumes and produces streams. A stream processing language is a domain-specific language designed for expressing streaming applications. The goal of a stream processing language is to strike a balance between the three requirements of *performance*, *generality*, and *productivity*. Performance is about answering high-throughput input streams with low-latency output streams. Generality is about making it possible to handle a variety of processing needs and data formats. And productivity is about enabling developers to write good code quickly.

Traditionally, programming languages have been characterized by their paradigm, including imperative, functional, declarative, object-oriented, etc. However, for streaming languages, the paradigm is not the most important characteristic; most streaming languages are more-or-less declarative. More important characteristics include the data model (e.g., relational, XML, RDF), execution model (e.g., synchronous, big-data), and target domain and users (e.g., event detection, reasoning, end-users). Section 2 surveys languages based on these characteristics. Section 3 generalizes from individual languages to extract recurring concepts and principles. Section 4 does the inverse: instead of looking at what most streaming languages have in common, it explores what most streaming languages lack. Finally, Section 5 concludes our paper.

2. STREAM PROCESSING LANGUAGES

There is much diversity in stream processing languages, stemming from different primary objectives, data models, and ways of thinking. This section surveys eight styles of stream processing languages. Each subsection introduces one of these styles using an exemplary language, followed by a brief discussion of important other languages of the same style.

2.1 Relational Streaming

```
1 Select IStream( Max(len) As mxl,
2                 MaxCount(len) As num,
3                 ArgMax(len, caller ) As who )
4 From Calls[Range 24 Hours Slide 1 Minute]
```

Figure 1: CQL code example.

In 2004, Arasu et al. at Stanford introduced CQL (for Continuous Query Language) [11]. CQL has been designed as an SQL-based declarative language for implementing continuous queries against streams of data, such as the LinearRoad benchmark [10]. The design was influenced by the TelegraphCQ system, which proposed an SQL-based language with a focus on expressive windowing constructs [29]. Figure 1 illustrates a CQL code example that uses a time-based sliding window (per minute within the last 24 hours) over phone calls to return the maximum phone call length along with its count and caller information.

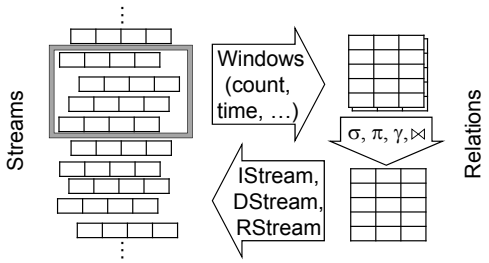


Figure 2: CQL algebra operators.

The semantics of CQL are based on two phases of data, *streams* and *relations*. As Figure 2 illustrates, CQL supports three classes of operators over these types. First, *stream-to-relation* operators freeze a stream into a relation. These operators are based on *windows* that, at any point of time, contain a historical snapshot of a recent portion of the stream. CQL includes time-based and tuple-based windows, both with optional partitioning. Second, *relation-to-relation* operators turn relations into another relation. These operators are expressed using standard SQL syntax and come from traditional relational algebra, such as select (σ), project (π), group-by-aggregate (γ), and join ($\bowtie$). Third, *relation-to-stream* operators thaw a relation back into a stream. CQL supports three operators of this class: `IStream`, `DStream`, and `RStream` (to capture inserts, deletes, or the entire relation).

Streaming SQL dialects were preceded by temporal relational models such as the one by Jensen and Snodgrass in the early '90s [55]. In their model, each temporal relation has two main dimensions: a valid time record and transaction time. Besides TelegraphCQ, another CQL predecessor was GSQL [35]. In addition to the standard SQL operators (e.g., σ, π, γ, $\bowtie$), GSQL supported a *merge* operator that combines streams from multiple sources in order as specified by ordered attributes. GSQL supported joins as long as it could determine a window from ordered attributes and join predicates.

CQL has influenced the design of many systems, for example, StreamInsight [5] and StreamBase [76]. Jain et al. described an approach to unify two different proposed SQL extensions for streams [54]. The first, by Oracle, was CQL-based and used a time-based execution model that could model simultaneity. The second, by StreamBase, used a tuple-based execution model that provided a way to react to primitive events as soon as they are seen by the system. SECRET goes beyond Jain et al.'s work to comprehensively understand the results of various window-based queries (e.g., time- and tuple-based windows) [19]. Zou et al. showed how to turn a stream of queries into a stream query by streaming their parameters [89]. Chandramouli et al. presented TiMR, which implemented temporal queries over the MapReduce framework [27]. And finally, Soulé et al. [77] presented a type system and small-step operational semantics for CQL via translation to the Brooklet stream-processing calculus [78].

2.2 Synchronous Dataflow

```
1 node tracker (speed, limit : int) returns (t: int);
2 var x: bool; cpt: int when x;
3 let
4   x = (speed > limit);
5   cpt = counter((0, 1) when x);
6   t = current(cpt);
7 tel
```

Figure 3: Lustre code example.

Synchronous dataflow (SDF) languages were introduced to ease the design of real-time embedded systems. They allow programmers to write a well-defined deterministic specification of the system. It is then possible to test, verify, and generate embedded code. The first dataflow synchronous languages Lustre [25] (Caspi and Halbwachs) and Signal [60] (Le Guernic, Benveniste, and Gautier) were proposed in France in the late 1980s. A dataflow synchronous program is a set of equations defining streams of values. Time proceeds by discrete logical steps, and at each step, the program computes the value of each stream depending on its inputs and possibly previously computed values. This approach is reminiscent of block diagrams, a popular notation to describe control systems. Figure 3 presents a Lustre code example that tracks the num-

ber of times the speed of a vehicle exceeds the speed limit. The counter cpt starts with 0 and is incremented by 1 each time the current speed exceed the current limit (when x). The return value t maintains the last computed value of cpt between two occurrences of x (current(cpt)).

Compared to the other languages presented here, SDF languages are relatively low level and target embedded controllers. The focus is on compiling efficient code that executes in bounded memory with a predictable execution time. In particular, this imposes that the schedule and communication rates can be statically computed by the compiler. Additional static analyses reject programs with potential initialization or causality issues. Compilers produce imperative code that can be executed in a control loop without communication buffers triggered by external events or on a periodic signal (e.g., every millisecond). The link between logical and real time is left to the designer of the system.

The dataflow synchronous approach has inspired multiple languages: Lucid Synchrone [73] combines the dataflow synchronous approach with functional features à la ML, StreamIt [80] focuses on efficient parallel processing of large streaming applications, and Zélus [20] is a Lustre-like language extended with ordinary differential equations to define continuous-time dynamics. Lustre is also the backbone of the industrial language and compiler Scade [34] routinely used to program embedded controllers in many critical applications.

2.3 Big-Data Streaming

```
1 stream<float64 len, rstring caller > Calls = CallsSrc() {}
2 type Stat = tuple<float64 len, int32 num, rstring who>;
3 stream<Stat> Stats = Aggregate(Calls) {
4     window Calls: sliding , time(24.0*60.0*60.0), time(60.0);
5     output Stats: len = Max(Calls.len),
6                   num = MaxCount(Calls.len),
7                   who = ArgMax(Calls.len, Calls. caller );
8 }
```

Figure 4: SPL code example.

The need to handle diverse data and processing requirements at scale motivated several recent big-data streaming languages and systems [3, 4, 24, 26, 51, 59, 69, 81, 87]. Each of them makes it easy to integrate operators written in general-purpose languages and to parallelize them on clusters of multicore computers. Hirzel et al. introduced the SPL language as part of the IBM Streams product in 2010 [50, 51]. Figure 4 shows an example for a similar use-case as Figure 1. Line 1 defines a stream Calls by invoking an operator CallsSrc, and Lines 3-8 define a stream Stats by invoking an operator Aggregate. An SPL program explicitly specifies a directed graph of stream edges and operator nodes. Streams carry tuples; in the examples, tuple attributes contain primitive values, but in general, they can also contain compound values such as other tuples or lists. Operators create and transform streams; operators are defined by users or libraries, not built into the language. Operators can be further configured upon invocation, for example, with windows or output assignments. To facilitate distribution, SPL's semantics are defined to require minimal synchronization between operators [77].

Like SPL, the core concept of other languages for big-data streaming is also that of a directed graph of streams and operators. This graph is an evolution of the query plan of earlier stream-relational systems. In fact, one can view Aurora [2], Borealis [1], and Spade [47] as the evolutionary links between relational and big-data streaming languages. They still focused on relational operators while already encouraging developers to explicitly code graphs.

Unlike SPL, which is a stand-alone language, later big-data streaming systems offer languages that are embedded in a general-purpose host language, typically Java. MillWheel focused on key-based partitioned parallelism and semi-automatic handling of out-of-order data [3]. Naiad focused on supporting both streaming and iterative batch analytics [69], using elaborate timestamps and a LINQ-based surface language [66]. Spark Streaming emulated streaming by repeated computations on immutable in-memory data batches [87]. Storm offered at-least-once semantics via buffering and acknowledgements [81]. Trill used batching to improve throughput and offered an extensible aggregation framework [26]. Heron displaced Storm by adding several improvements, such as a back-pressure mechanism [59]. Beam picks up where MillWheel left off, giving programmers ways to reconcile event time and processing time [4]. And finally, Flink focuses on supporting both real-time streaming and batch analytics [24].

All of the above-listed big-data streaming systems offer embedded languages for specifying more-or-less explicit stream graphs. An embedded language is an advanced library or framework that makes heavy use of host-language abstractions such as lambdas, generics, and local variable type inference. For instance, LINQ integrates SQL-inspired query syntax in a general-purpose language [66]. Embedded languages offer simple interoperability with their host language, as well as leveraging host-language tools and skills [52]. On the downside, since they are not self-contained, they are hard to isolate clearly from the host language, inhibiting debugging, optimization, and standardization.

2.4 Complex Event Processing

```
1  stream<Alert> Alerts = MatchRegex(Calls) {
2    param
3      partitionBy : caller ;
4      predicates : {
5        tooFarTooFast =
6              geoDist( First (loc ), Last(loc )) >= 10.0
7          && timeDist(First(ts ), Last(ts )) <= 60.0; };
8      pattern     : ".+ tooFarTooFast";
9    output
10     Alerts : who=caller, where=Last(loc), when=Last(ts);
11 }
```

Figure 5: CEP example.

Complex event processing (CEP) uses patterns over simple events to detect higher-level, *complex*, events that may comprise multiple simple events. CEP can be considered either as an alternative to stream processing or as a special case of stream processing. The latter consideration has led to the definition of CEP operators in streaming languages. For example, the MatchRegex [49] operator implements CEP in the library of the SPL language [51] (Section 2.3). MatchRegex was introduced by Hirzel in 2012, influenced by the MATCH-RECOGNIZE proposal for extending ANSI SQL [88]. Compared to its SQL counterpart, MatchRegex is simplified, syntactically concise, and easy to deploy as a library operator. MatchRegex is implemented via code generation and translates to an automaton for space- and time-efficient incremental computation of aggregates. However, it omits other functionalities beyond pattern matching, such as joins and reporting tasks. Figure 5 shows an example for detecting a complex event when simple phone-call events occur over 10 miles apart within 60 seconds. Line 8 defines the regular expression, where the period (.) matches any simple event; the plus (+) indicates at-least-once repetition; and tooFarTooFast is a simple event defined via a predicate in Lines 5–7. The First and Last functions reference corresponding simple events in the overall match: in this case, the start of the sequence matched by .+ and the simple event matched by tooFarTooFast.

One of the earliest languages for complex event queries on real-time streams was SASE [86]. The language was designed to translate to algebraic operators, but did not yet support aggregation or regular expressions with Kleene closure, as used in Figure 5. The Cayuga Event Language offered aggregation and Kleene closure, but did so in a handcrafted syntax instead of familiar regular expression syntax [42]. The closest predecessor of MatchRegex was the MATCH-RECOGNIZE proposal for extending SQL with pattern recognition in relational rows [88]. It used regular-expression syntax as well as aggregations. Like MatchRegex, it is embedded in a host language that supports orthogonal features via operators such as joins. Another take on CEP using regular expressions was EventScript [33], which allowed the patterns to be interspersed with action blocks. While most CEP pattern matching is inherently sequential, Chandramouli et al. generalized it for out-of-order data streams [28], a topic further discussed in Section 4.1.

Recently, CEP is also supported by several big-data streaming engines, such as Trill [26], Esper [45], and Flink [24], the latter exhibiting a CEP library since its early 1.0 version. Indeed, the high throughput and low latency nature of these engines make them suitable for CEP's real time analytics.

2.5 XML Streaming

In 2002, Diao et al. [44] presented YFilter, which implemented continuous queries over XML streaming data using a subset of the XPath language [32]. YFilter applied a multi-query optimization that used a single finite state machine to represent and evaluate several XPath expressions. In particular, YFilter exploited commonalities among path queries by merging the common prefixes of the paths so that they were processed at most once. This shared processing improved performance significantly by avoiding redundant processing for duplicate path expressions.

Before YFilter, which processed streams of XML documents, came NiagaraCQ, which processed update streams to existing XML documents [30], borrowing syntax from XML-QL [43]. NiagaraCQ supported incremental evaluation to consider only the changed portion of each updated XML file. It supported two kinds of continuous queries: *change-based* queries, which trigger as soon as new relevant data becomes available, and *timer-based* queries, which trigger only at specified time intervals. XSQ is an XPath-based language for not just filtering but transforming streams of XML documents [71]. And XMLParse is an operator for XML stream transformation in a big-data streaming language [67].

2.6 RDF Streaming

In 2009, Della Valle et al. called the semantic web and AI community to investigate how to represent, manage, and reason on heterogeneous data streams in the presence of expressive domain models (captured by ontologies) [38]. Those communities were still focusing on static knowledge bases, and solutions to incorporate changes were too complex to apply to big data streams. Della Valle et al. pro-

posed to name this new research area *stream reasoning* [41], and the sub-area focused on the semantic web *RDF Stream Processing* (RSP) [82]. This section presents RSP, while Section 2.7 elaborates on stream reasoning.

RSP research extended the semantic web stack [8] to represent heterogeneous streams, continuous queries, and continuous reasoning. Inspired by CQL [11], Della Valle et al. proposed Continuous SPARQL (C-SPARQL, [37]), inspiring multiple extensions [7, 23, 61]. In 2013, a W3C community group[1] was established to define RSP-QL syntax [39] and semantics [40]. In RSP-QL, an *RDF stream* is an unbounded sequence of time-varying graphs $\langle t, \tau \rangle$, where t is an RDF graph and τ is a non-decreasing timestamp. A RSP-QL query is a continuous query on multiple RDF streams and graphs.

```
1  REGISTER STREAM :out
2  AS CONSTRUCT RSTREAM { ?x a :Hub }
3  FROM NAMED WINDOW :lwin
4         ON :in [ RANGE PT120M STEP PT10M]
5  FROM NAMED WINDOW :swin
6         ON :in [ RANGE PT10M STEP PT10M]
7  WHERE {
8    WINDOW :lwin{
9      SELECT ?x ( COUNT(*) AS ?totalLong)
10     WHERE { ?c1 :callee ?x. }
11     GROUP BY ?x }
12   WINDOW :swin{
13     SELECT ?x ( COUNT(*) AS ?totalShort)
14     WHERE { ?c2 :callee ?x. }
15     GROUP BY ?x }
16   GRAPH :bg {?x :hasStandardDeviation ?s }
17   FILTER ((?totalShort − ?totalLong/12)/?s > 2)
18 } GROUP BY ?x
```

Figure 6: RSP-QL example.

Figure 6 illustrates an RSP-QL query that continuously identifies *communication hubs*. The idea is to find callees who appear more frequently than usual. Line 1 registers stream out and Line 2 sends the query result on that stream. Lines 3-6 open a short 10-minute tumbling window swin and a long 2-hour sliding window lwin on the input stream in. Lines 8-11 and 12-15 count the number of calls per callee in the long and short window, respectively. Lines 16-17 fetch the standard deviation of the number of calls for each callee from a static graph, join it with the callees appearing in both windows, and select callees two standard deviations above average.

2.7 Stream Reasoning

Automated reasoning plays a key role in modern information integration where an ontology offers a conceptual view over pre-existing autonomous data sources [63]. In this setting, the reasoner can find answers that are not syntactically present in the data sources, but are deduced from the data and the ontology. This query-answering approach is called ontology-based data access [72].

[1] http://www.w3.org/community/rsp/

```
1  SubObjectPropertyOf(
2     ObjectPropertyChain( :calls :calls ) :gossips
3  )
4  TransitiveObjectProperty( :gossips )
5
6  REGISTER STREAM GossipMeter AS
7  SELECT (count(?x) AS ?impact)
8  FROM NAMED WINDOW :win
9         ON :in [ RANGE PT60M STEP PT10M]
10 WHERE { :Alice :gossips ?x }
```

Figure 7: Stream reasoning example with two ontological axioms and a RSP-QL query.

As RDF is the dominant data model in reasoning for data integration, RDF streaming languages (Section 2.6) bridge the gap between stream processing and ontology-based data integration. Della Valle et al. opened up this direction, showing how continuous reasoning can be reduced to periodic repetition of reasoning over a windowed ontology stream [37]. Figure 7 shows an RSP-QL query that uses reasoning to continuously count how many people :Alice gossips with. Consider an RDF stream with the triples $\langle$:Alice :calls :Bob, $\tau_i\rangle$ and $\langle$:Bob :calls :Carl, $\tau_{i+1}\rangle$. Lines 1-4 define :gossips as the transitive closure of :calls. When the window contains these two triples, the RSP-QL query returns 2, because :Alice :gossips :Bob directly calling him, but the system can also deduce that she :gossips :Carl indirectly via :Bob.

While conceptually simple, this kind of reasoning is hard to do efficiently. Barbieri et al. [14] and Komazec et al. [57] pioneered it optimizing the DRed algorithm observing that in stream processing deletion becomes predictable. The current state-of-the-art is the work of Motik et al. [68].

In parallel, Ren and Pan proposed an alternative approach via truth maintenance systems [74]. Calbimonte et al. exploited ontology-based data access [22]. Heintz et al. developed logic-based spatio-temporal stream reasoning [36]. Anicic et al. bridged stream reasoning with complex event processing grounding both in logic programming [7]. Beck et al. used answer set programming to model expressive stream reasoning tasks [17] in Ticker [18] and Laser [16]. Inductive stream reasoning, i.e., applying machine-learning to RDF streams or to ontology streams, is also an active field [15, 31, 62].

2.8 Streaming for End-Users

We use the term *end-users* to refer to users without particular software development training. Prob-

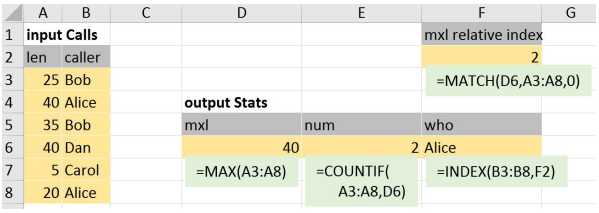


Figure 8: ActiveSheets example.

ably the most successful programming tool for end-users is spreadsheet formulas. And from the early days of VisiCalc in 1979 [21], spreadsheet formulas have been *reactive* in the sense that any changes in their inputs trigger an automatic recomputation of their outputs. Therefore, in 2014, Vaziri et al. designed ActiveSheets, a spreadsheet-based stream programming model [84]. Figure 8 gives an example that implements a similar computation as Figure 1. Cells A3:B8 contain a sliding window of recent call records, which ActiveSheets updates from live input data. Cells D6:F6 contain the output data, (re-)computed using reactive spreadsheet formulas. The formula E6=COUNTIF(A3:A8,D6) counts how many calls in the window are as long as a longest call. The formula F6=INDEX(B3:B8,F2) uses the relative index F2 of the longest len to retrieve the corresponding caller. ActiveSheets was influenced by synchronous dataflow, discussed in Section 2.2. Of course, spreadsheets are not the only approach for end-user programming. For instance, MARIO constructed streaming applications automatically based on search terms [75]. Linehan et al. used a controlled natural language for specifying event processing rules [65]. And TEM used model-driven development based on a spreadsheet [46].

3. PRINCIPLES

The previous section described concrete stream processing languages belonging to several families. This section takes a cross-cutting view and explores concepts that many of these languages have in common by identifying the language design principles behind the concepts. The views and opinions expressed herein are those of the authors and are not meant as the final word. Explicitly articulating principles demystifies the art of language design. We categorize language design principles according to the three requirements from Section 1, namely performance, generality, and productivity.

The **performance** requirement is addressed by streaming language design principles $\mathbf{P_1}$–$\mathbf{P_4}$:

$\mathbf{P_1}$ *Windowing principle.* Windows turn streaming data into static data suitable for optimized static computation. For instance, in CQL, windows produce relations suitable for classic relational algebra [9], optimizable via classic relational rewrite rules (see Figure 2).

$\mathbf{P_2}$ *Partitioning principle.* Key-based partitions enable independent computation over disjoint state, thus simplifying data parallelism. For instance, MatchRegex performs complex event processing separately by partition [49] (see Line 3 of Figure 5). Principles P_1 and P_2 also simplify advanced state management, e.g., in key-value stores for operator migration [48].

$\mathbf{P_3}$ *Stream graph principle.* Streaming applications are graphs of operators that communicate almost exclusively via streams, making them easy to place on different cores or machines. This principle is central to the big-data languages in Section 2.3 such as SPL [51] (see Figure 4).

$\mathbf{P_4}$ *Restriction principle.* The schedules and communication rates in a streaming application are restricted for both performance and safety. For instance, Lustre can be compiled to a simple imperative control loop without communication buffers [25] (see Section 2.2).

The **generality** requirement is addressed by streaming language design principles $\mathbf{P_5}$–$\mathbf{P_8}$:

$\mathbf{P_5}$ *Orthogonality principle.* Basic language features are irredundant and work the same independently of how they are composed. For instance, in CQL, relational-algebra operators are orthogonal to windows [9] (see Section 2.1).

$\mathbf{P_6}$ *No-built-ins principle.* The core language remains slim and regular by enabling extensions in the library. For instance, in SPL, relational operators are not built into the language, but are user-defined in the library instead [51] (see Lines 3–8 of Figure 4).

$\mathbf{P_7}$ *Auto-update principle.* The syntax of conventional non-streaming computation is overloaded to also support reactive computation. For instance, ActiveSheets uses conventional spreadsheet formulas, updating their output when input cells change [84] (see Figure 8). The Lambda or Kappa architectures [58] take this to the extreme by combining batch and streaming outside of the language.

$\mathbf{P_8}$ *General-feature principle.* Similar special-case features are replaced by a single more-general feature. For instance, operator parameters in SPL [51] accept general uninterpreted expressions, including predicates for the special case of CEP [49] (see Lines 4–7 of Figure 5).

The **productivity** requirement is addressed by streaming language design principles $\mathbf{P_9}$–$\mathbf{P_{12}}$:

Language	*Performance*	*Generality*	*Productivity*
CQL	P_1 P_2 P_3	**P_5** P_8	**P_9**
Lustre	P_4	**P_5** P_6 P_7 P_8	**P_9** P_{10} P_{11} P_{12}
SPL	P_1 P_2 P_3	**P_5** P_6 P_8	**P_9** P_{11} P_{12}
MatchRegex	P_2	**P_5** P_6 P_8	**P_9** P_{10} P_{12}
YFilter	P_4	**P_5** P_6	**P_9** P_{10}
RSP-QL	P_1 P_3	**P_5** P_6 P_8	**P_9** P_{10} P_{11}
ActiveSheets	P_1 P_2 P_4	**P_5** P_6 P_7 P_8	**P_9** P_{10}

Table 1: Which of the languages that served as examples in Section 2 satisfy which of the language design principles in Section 3.

P_9 *Familiarity principle.* The syntax of non-streaming features in streaming languages is the same as in non-streaming languages. This makes the streaming language easier to learn. For instance, CQL [11] adopts the select-from-where syntax of SQL (see Figure 1).

P_{10} *Conciseness principle.* The most concise syntax is reserved for the most common tasks. This increases productivity since there is less code to write and read. For instance, regular expressions represent "followed-by" concisely via juxtaposition $e_1\, e_2$ (see Line 8 of Figure 5).

P_{11} *Regularity principle.* Data literals, patterns that match them, and/or declarations all use similar syntax. For instance, RSP-QL uses pattern syntax resembling concrete RDF triples (see Line 10 of Figure 6).

P_{12} *Backward reference principle.* Code direction is consistent with both scope and control dominance, for readability. For example, Lustre declares variables before their use (see Figure 3).

3.1 Principles Summary

Good language design is driven by principles, but it is also an exercise in prioritizing among these principles. For instance, CQL satisfies P_9 (familiarity principle) by adopting SQL's syntax and CQL violates P_{12} (backward reference principle) by adopting SQL's scoping rules. Table 1 summarizes principles by language. Only two of the twelve principles (P_5 and P_9, shown in bold) are uniformly covered, both related to the ease of use of the language (separation of concerns and syntax familiarity). Although some of the languages exhibit fewer principles, Table 1 does not provide a comparative metric for quantifying the coverage of each principle; such a metric would be hard to agree upon. Satisfying more principles does not automatically imply satisfying the associated requirement better. While we formulated the principles from the perspective of streaming languages, we do not claim to have invented them: many are well-known from the design of other programming languages. For instance, the orthogonality principle was a stated aim of the Algol 68 language specification [83]. Now that we have seen concepts that are *present* in most streaming languages, the next section will explore what is commonly *missing or underdeveloped*.

4. WHAT'S NEXT?

In the Big Data era, the need to process and analyze a high volume of data is a fundamental problem. Industry analysts point out that besides volume, there are also challenges in variety, velocity, and veracity [53]. Streaming languages naturally handle volume and velocity of the data, since they are designed to process data in real-time in a streaming way. Thus, in the following, we focus on veracity and variety, since there are more open research challenges in these directions despite much recent progress in streaming languages. In addition, we elaborate on the challenge of adoption, which is an important problem of programming languages in general and of streaming languages in particular.

4.1 Veracity

With the evolution of the internet of things and related technologies, many end-user applications require stream processing and analytics. Streaming languages should ensure veracity of the output stream in terms of accuracy, correctness, and completeness of the results. Furthermore, they should not sacrifice performance either, answering high-throughput input streams with low-latency output streams. Veracity in a streaming environment depends on the semantics of the language since the stream is infinite and new results may be added or computed aggregates may change. It is important that the output stream for a given input stream be well-defined based on the streaming language semantics. For example, if the language offers a sliding time window feature, any aggregate should be computed correctly at any time point based on all data within the time window. Stream veracity problems may occur for different reasons. For example, in multi-streaming applications, each stream may be produced by sensors. Errors may occur either in the data itself (e.g., noisy sensor readings) or by delays or data loss during the transfer to the stream processing system. For instance, data may arrive out-of-order because of communication delays or because of the inevitable time drift between independent distributed stream sources. Ideally, the output stream should be accurate, complete, and timely even if errors occur in the input stream. Unfortunately, this is not always feasible.

Why is this important? Veracity of the output of streaming applications is important when high-stakes and irreversible decisions are based on these outputs. In the big-data era, veracity is one of the most important problems even for non-streaming data processing, and stream processing makes veracity even more challenging than in the static case. Streams are dynamic and usually operate in a distributed environment with minimal control over the underlying infrastructure. Such a loosely coupled model can lead to situations where any data source can join and leave on the fly. Moreover, stream-producing sensors have limitations such as processing power, energy level, or memory consumption, which can easily compromise veracity.

How can we measure the challenge? To estimate the robustness of a streaming language implementation to veracity problems, we define as *ground truth* the output stream in the absence of veracity problems (for example data loss or delayed data). Then we can quantify veracity. Let *error* be a function that compares the produced result of an approach with and without veracity problems. An example of an error function is the number of false positives and false negatives. An approach is *robust* for veracity of streaming data if the error scales at most linearly with respect to the size and the error rate of the input stream, while the delay in the latency is bounded and independent of the input size. The streaming language veracity challenge can be broken down into the following measures $\mathbf{C_1}$–$\mathbf{C_3}$:

- $\mathbf{C_1}$ *Fault-tolerance.* A program in the language is robust even if some of its components fail. The language can define different behaviors, for example, at-least-once semantics in Storm [81] or check-pointing in Spark Streaming [87].
- $\mathbf{C_2}$ *Out-of-order handling.* This measure has two facets. First, the streaming language should have clear semantics about the expected result. Second, the streaming language should be robust to out-of-order data and should ensure that the expected output stream is produced with limited latency. Li et al. define out-of-order stream semantics based on low watermarks [64]; Spark Streaming relies on idempotence to handle stragglers [87]; and Beam separates event time from processing time [4].
- $\mathbf{C_3}$ *Inaccurate value handling.* A program in the language is robust even if some of its input data is wrong. The language can help by supporting statistical quality measures [85].

Why is this difficult? In stream processing, data is typically sent on a best-effort basis. As a result, data can be lost, incorrect, arrive out of order, or be approximate. This is exacerbated by the fact that the streaming setting affords limited opportunity to compensate for these issues. Furthermore, the performance requirements of streaming systems encourage the use of approximate computing [12], thus increasing the uncertainty of the data. Also, machine-learning often yields uncertain results due to imperfect generalization. An important aspect of streaming data is ordering, typically characterized by time. The correctness of the response to queries depends on the source of ordering, such as the creation, processing, or delivery time. Stream processing often requires that each piece of data must be processed within a window, which can be characterized by predefined size or temporal constraints. In stream settings, sources typically do not receive control feedback. Consequently, when exceptions occur, recovery must occur at the destination. This reduces the space of possibilities for handling transaction rollbacks and fault tolerance.

4.2 Data Variety

Data variety refers to the presence of different data formats, data types, data semantics, and associated data management solutions in an information system. The term emerged with the advent of Big Data, but the problem of taming variety is well known for machine understanding of unstructured data such as text, images, and video as well as (syntactic, structural, and semantic) interoperability and data integration for structured and semistructured data. There are multiple known solutions to data variety for a moderate number of high-volume data sources. But data variety is still unsolved when there are hundreds of data sources to integrate or when the data to integrate is highly dynamic or streaming (as in this paper).

Why is this important? Increasingly, applications must process heterogeneous data streams in real-time together with large background knowledge bases. Consider the following two examples from [41] (where interested readers can find others).

In the first example, we want to use sensor readings of the last 10 minutes to find electricity-producing turbines that are in a state similar (e.g., Pearson correlated by at least 0.75) to any turbine that subsequently had a critical failure. Here, data variety arises from having tens of turbines of 3-4 different types equipped with different sensors deployed other many years, where more sensors will be deployed in the future. Moreover, in many cases, once an anomaly is detected, the user also needs to retrieve multimedia maintenance instructions and

annotations to complete the diagnosis process.

In the second example, we want to use the latest open traffic information and social media as well as the weather forecast to determine if the users of a mobile mobility app are likely to run into a traffic jam during their commute tonight and how long it will take them to get home. Here, data variety arises from using third-party data sources that are free to evolve in syntax, structure, and semantics.

How can we measure the challenge? The streaming language data variety challenge can be broken down into the following measures $\mathbf{C_4}$–$\mathbf{C_6}$:

- $\mathbf{C_4}$ *Expressive data model.* The data model used to logically represent information is expressive and allows encoding multiple data types, data structures, and data semantics. This is the path investigated by RSP-QL [41, 82].
- $\mathbf{C_5}$ *Multiple representations.* The language can ingest data in multiple representations, offering the programmer a unified set of logical operators while implementing physical operators that work directly on the representations for performance. An example is the most recent evolution of the Streaming Linked Data framework [13].
- $\mathbf{C_6}$ *New sources with new formats.* The language allows adding new sources where data are represented in a format unforeseen when the language was released. This might be accomplished by extending R2RML[2].

Why is this difficult? Deriving value is harder for a system that has to tame data variety than for a system that only has to handle a single well-structured data source. This is because solutions that analyze data require homogeneous well-formed input data, so, when there is data variety, preparing such data requires a number of different data management solutions that take time to perform their part of the processing as well as to coordinate among each others. This time is particularly relevant in stream processing, where answers should be generated with low latency. Even if the time available to answer depends on the application domain (in call centers, routing needs to be decided in subseconds, while in oil operations, dangerous situations must be detected within minutes), traditional batch pipelines for feature extraction and extract-transform-load (ETL) may take so long that the results, when computed, are no longer useful. For this reason, it is still challenging to tame variety in stream processing systems.

[2] `https://www.w3.org/TR/r2rml`

4.3 Adoption

Stream processing languages have an adoption problem. As Section 2 illustrates, there are several families of streaming languages comprising several members each. But no one streaming language has been broadly adopted. The language family receiving the most attention from large technology companies is big-data streaming, including offerings by Google [3], Microsoft [5], IBM [51], and Twitter [81]. However, they all differ. Furthermore, in the pursuit of interoperability and expediency, most big-data streaming languages are not stand-alone but embedded in a host language. While being embedded gives a short-term boost to language development, the entanglement with a host language makes it hard to offer stable and clear semantics. And, if the history of databases is any guide, such stable and clear semantics are useful for agreeing on and consistently implementing a standard. Part of the reason that the relational model for databases displaced its disparate predecessors is its strong mathematical foundation. One of the most-used languages mentioned in this survey is Scade [34], but it is designed for embedded systems and not big-data streaming. Getting broad adoption for a big-data streaming language remains an open challenge.

Why is this important? Solving the adoption problem for stream processing languages would yield many benefits. It would encourage students to build marketable skills and give employers a sustainable hiring pipeline. It would raise attention to streaming innovation, benefiting researchers, and to streaming products, benefiting vendors. If most systems adopted more-or-less the same language, they would become easier to benchmark against each other. Other popular programming languages, such as SQL, Java, and JavaScript, flourished when companies competed against each other to provide better implementations of the language. On the downside, focusing on a single language would reduce the diversity of the eco-system, transforming innovation and competition from being broad to being deep. But overall, if the problem of streaming language adoption were solved, we would expect streaming systems to become more robust and faster.

How can we measure the challenge? The streaming language adoption challenge can be broken down into the following measures $\mathbf{C_7}$–$\mathbf{C_9}$:

- $\mathbf{C_7}$ *Widely-used implementation of one language.* One language in the family has at least one implementation that is widely used in practice, for instance, Scade for SDF [34].
- $\mathbf{C_8}$ *Standard proposal or standard.* There are serious efforts towards an official standard, for

Languages	*Veracity*	*Variety*	*Adoption*
Relational	**C2**		**C8** **C9**
Synchronous		**C4**	C7 **C9**
Big Data	C1 **C2**	**C4** C5 C6	C7
CEP	**C2**	**C4**	**C8**
XML		**C4** C6	
Stream Reasoning	C3	**C4** C5 C6	**C8** **C9**
End-user		**C4**	

Table 2: Which of the language families from Section 2 address which of the measures of streaming language challenges in Section 4.

instance, Jain et al. for StreamSQL [54] or MATCH-RECOGNIZE for CEP [88].

C9 *Multiple implementations of same language.* One language in the family has multiple more-or-less compatible implementations, for instance, Lustre [25] and Scade [34] for SDF.

Language adoption is driven not just by the technical merits of the language itself but also by external factors, such as industry support or implementations that are open-source with open governance.

Why is this difficult? Adoption is hard for any programming language, but particularly so for a streaming language. While streaming in general is not new [79], big-data streaming is a relatively recent phenomenon. And big-data streaming, in turn, is driven by several ongoing industry trends, including the internet of things, cloud computing, and artificial intelligence (AI). Since all three of these trends are themselves actively shifting, they provide an unstable ecosystem for streaming languages to evolve. Furthermore, innovation often takes place in a setting where data is assumed to be at rest, as opposed to streaming, where data is in motion. For instance, most AI algorithms work over a fixed training data set, so additional research is necessary to make them work well online. When it comes to streaming languages, there is not even a consensus on what are the most important features to include. For instance, both the veracity and the variety challenge discussed previously have given rise to many feature ideas that have yet to make it into the mainstream. Since people come to streaming research from different perspectives, they sometimes do not even know each other's work, inhibiting adoption. This survey aims to mitigate that problem.

4.4 Challenges Summary

Table 2 summarizes the challenges. Compared to the coverage of principles in Table 1, the coverage of challenges is more sparse and spread out over research prototypes. That is why we tabulated it for language families instead of individual languages. There is much space for future work. The measures highlighted in bold are most covered across all the languages families. The ability to handle a wide variety of data formats appears to be a universal concern. Ultimately, we aim at streaming languages that are both principled and close the gap on all challenges.

5. CONCLUSION

This paper surveys recent stream processing languages. Given their numbers, it appears likely that more will be invented soon. We hope this survey will help the field evolve towards better languages by helping readers understand the state of the art.

Acknowledgements

The idea for this paper was conceived at Dagstuhl Seminar 17441 on "Big Stream Processing Systems". We thank our fellow participants of the seminar and the Dagstuhl staff. A. Bonifati's work was supported by CNRS Mastodons MedClean. S. Sakr's work was funded by the European Regional Development Fund via the Mobilitas Pluss program (grant MOBTT75). A. Vlachou's work was supported by the datAcron project, funded through the European Union's Horizon 2020 research and innovation program (grant 687591).

6. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *Conference on Innovative Data Systems Research (CIDR)*, pages 277–289, 2005.

[2] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *Journal on Very Large Data Bases (VLDB J.)*, 12(2):120–139, Aug. 2003.

[3] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. MillWheel: Fault-tolerant stream processing at internet scale. In *Conference on Very Large Data Bases (VLDB) Industrial Track*, pages 734–746, Aug. 2013.

[4] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In *Conference on Very Large Data Bases (VLDB)*, pages 1792–1803, Aug. 2015.

[5] M. H. Ali, C. Gerea, B. Raman, B. Sezgin, T. Tarnavski, T. Verona, P. Wang, P. Zabback, A. Kirilov, A. Ananthanarayan, M. Lu, A. Raizman, R. Krishnan, R. Schindlauer, T. Grabs, S. Bjeletich, B. Chandramouli, J. Goldstein, S. Bhat, Y. Li, V. Di Nicola, X. Wang, D. Maier, I. Santos, O. Nano, and S. Grell. Microsoft CEP server and online behavioral targeting. In *Demo at Conference on Very Large Data Bases (VLDB-Demo)*, pages 1558–1561, 2009.

[6] H. C. Andrade, B. Gedik, and D. S. Turaga. *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press, 2014.

[7] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012.

[8] G. Antoniou, P. T. Groth, F. van Harmelen, and R. Hoekstra. *A Semantic Web Primer, 3rd Edition*. MIT Press, 2012.

[9] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *Journal on Very Large Data Bases (VLDB J.)*, 15(2):121–142, 2006.

[10] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *Conference on Very Large Data Bases (VLDB)*, pages 480–491, 2004.

[11] A. Arasu and J. Widom. A denotational semantics for continuous queries over streams and relations. *SIGMOD Record*, 33(3):6–11, 2004.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[13] M. Balduini, E. D. Valle, and R. Tommasini. SLD revolution: A cheaper, faster yet more accurate streaming linked data framework. In *European Semantic Web Conference (ESWC) Satellite Events*, pages 263–279, 2017.

[14] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *Extended Semantic Web Conference (ESWC)*, pages 1–15, 2010.

[15] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, Y. Huang, V. Tresp, A. Rettinger, and H. Wermser. Deductive and inductive stream reasoning for semantic social media analytics. *IEEE Intelligent Systems*, 25(6):32–41, 2010.

[16] H. R. Bazoobandi, H. Beck, and J. Urbani. Expressive stream reasoning with Laser. In *International Semantic Web Conference (ISWC)*, pages 87–103, 2017.

[17] H. Beck, M. Dao-Tran, T. Eiter, and M. Fink. LARS: A logic-based framework for analyzing reasoning over streams. In *Conference on Artificial Intelligence (AAAI)*, pages 1431–1438, 2015.

[18] H. Beck, T. Eiter, and C. Folie. Ticker: A system for incremental ASP-based stream reasoning. *Theory and Practice of Logic Programming (TPLP)*, 17(5-6):744–763, 2017.

[19] I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. J. Miller, and N. Tatbul. SECRET: A model for analysis of the execution semantics of stream processing systems. In *Conference on Very Large Data Bases (VLDB)*, pages 232–243, 2010.

[20] T. Bourke and M. Pouzet. Zélus: A synchronous language with ODEs. In *Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 113–118, 2013.

[21] D. Bricklin and B. Frankston. VisiCalc computer software program, 1979. Reference Manual, Personal Software Inc.

[22] J. Calbimonte, J. Mora, and Ó. Corcho. Query rewriting in RDF stream processing. In *European Semantic Web Conference (ESWC)*, pages 486–502, 2016.

[23] J.-P. Calbimonte, O. Corcho, and A. J. G. Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference (ISWC)*, pages 96–111, 2010.

[24] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink: Stream and batch processing in a single engine. *IEEE Database Engineering Bulletin*, 36(4):28–38, Dec. 2015.

[25] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: a declarative language for real-time programming. In *Symposium on Principles of Programming Languages (POPL)*, pages 178–188, 1987.

[26] B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, D. Fisher, J. C. Platt, J. F. Terwilliger, and J. Wernsing. Trill: A high-performance incremental query processor for diverse analytics. In *Conference on Very Large Data Bases (VLDB)*, pages 401–412, Aug. 2014.

[27] B. Chandramouli, J. Goldstein, and S. Duan. Temporal analytics on big data for web advertising. In *International Conference on Data Engineering (ICDE)*, pages 90–101, 2012.

[28] B. Chandramouli, J. Goldstein, and D. Maier. High-performance dynamic pattern matching over disordered streams. In *Conference on Very Large Data Bases (VLDB)*, pages 220–231, 2010.

[29] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Conference on Innovative Data Systems Research (CIDR)*, pages 668–668, 2003.

[30] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *International Conference on Management of Data (SIGMOD)*, pages 379–390, 2000.

[31] J. Chen, F. Lécué, J. Z. Pan, and H. Chen. Learning from ontology streams with semantic concept drift. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 957–963, 2017.

[32] J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C recommendation, W3C, Nov. 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[33] N. H. Cohen and K. T. Kalleberg. EventScript: An event-processing language based on regular expressions with actions. In *Conference on Languages, Compiler, and Tool Support for Embedded Systems (LCTES)*, pages 111–120, 2008.

[34] J.-L. Colaco, B. Pagano, and M. Pouzet. Scade 6: A formal language for embedded critical software development. In *Symposium on Theoretical Aspect of Software Engineering (TASE)*, 2017.

[35] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *International Conference on Management of Data (SIGMOD) Industrial Track*, pages 647–651, 2003.

[36] D. de Leng and F. Heintz. Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks. In *Conference on Artificial Intelligence (AAAI)*, pages 957–963, 2016.

[37] E. Della Valle, S. Ceri, D. F. Barbieri, D. Braga, and A. Campi. A first step towards stream reasoning. In *Future Internet Symposium (FIS)*, pages 72–81, 2008.

[38] E. Della Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a streaming world! Reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[39] D. Dell'Aglio, J. Calbimonte, E. Della Valle, and Ó. Corcho. Towards a unified language for RDF stream query processing. In *European Semantic Web Conference (ESWC) Satellite Events*, pages 353–363, 2015.

[40] D. Dell'Aglio, E. Della Valle, J. Calbimonte, and Ó. Corcho. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(4):17–44, 2014.

[41] D. DellAglio, E. Della Valle, F. van Harmelen, and A. Bernstein. Stream reasoning: A survey and outlook. *Data Science*, 1(1-2):59–83, 2017.

[42] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White. Cayuga: A general purpose event monitoring system. In *Conference on Innovative Data Systems Research (CIDR)*, pages 412–422, 2007.

[43] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. *Computer Networks*, 31(11):1155–1169, 1999.

[44] Y. Diao, P. M. Fischer, M. J. Franklin, and R. To. YFilter: Efficient and scalable filtering of XML documents. In *Demo at International Conference on Data Engineering (ICDE-Demo)*, pages 341–342, 2002.

[45] Esper. Esper open source software for streaming analytics, 2018. `http://www.espertech.com/esper/` (Retrieved June 2018).

[46] O. Etzion, F. Fournier, I. Skarbovsky, and B. von Halle. A model driven approach for event processing applications. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 81–92, 2016.

[47] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: The System S declarative stream processing engine. In *International Conference on Management of Data (SIGMOD)*, pages 1123–1134, 2008.

[48] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu. Elastic

scaling for data stream processing. *Transactions on Parallel and Distributed Systems (TPDS)*, 25(6):1447–1463, June 2014.
[49] M. Hirzel. Partition and compose: Parallel complex event processing. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 191–200, 2012.
[50] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, M. Mendell, H. Nasgaard, R. Soulé, and K.-L. Wu. SPL Streams Processing Language Specification. Technical Report RC24897, IBM Research, 2009.
[51] M. Hirzel, S. Schneider, and B. Gedik. SPL: An extensible language for distributed stream processing. *Transactions on Programming Languages and Systems (TOPLAS)*, 39(1):5:1–5:39, March 2017.
[52] P. Hudak. Modular domain specific languages and tools. In *International Conference on Software Reuse (ICSR)*, pages 134–142, 1998.
[53] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM (CACM)*, 57(7):86–94, July 2014.
[54] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Cetintemel, M. Cherniack, R. Tibbets, and S. Zdonik. Towards a streaming SQL standard. In *Conference on Very Large Data Bases (VLDB)*, pages 1379–1390, 2008.
[55] C. S. Jensen and R. Snodgrass. Temporal specialization and generalization. *Transactions on Knowledge and Data Engineering (TKDE)*, 6(6):954–974, 1994.
[56] W. M. Johnston, J. R. P. Hanna, and R. J. Millar. Advances in dataflow programming languages. *ACM Computing Surveys (CSUR)*, 36(1):1–34, 2004.
[57] S. Komazec, D. Cerri, and D. Fensel. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 58–68, 2012.
[58] J. Kreps. Questioning the lambda architecture, 2014. `http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html` (Retrieved June 2018).
[59] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter Heron: Stream processing at scale. In *International Conference on Management of Data (SIGMOD)*, pages 239–250, May 2015.
[60] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with Signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.
[61] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, and M. Fink. Linked stream data processing engines: Facts and figures. In *International Semantic Web Conference (ISWC)*, pages 300–312, 2012.
[62] F. Lécué and J. Z. Pan. Predicting knowledge in an ontology stream. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2662–2669, 2013.
[63] M. Lenzerini. Data integration: A theoretical perspective. In *Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.
[64] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier. Out-of-order processing: A new architecture for high-performance stream systems. In *Conference on Very Large Data Bases (VLDB)*, pages 274–288, 2008.
[65] M. H. Linehan, S. Dehors, E. Rabinovich, and F. Fournier. Controlled English language for production and event processing rules. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 149–158, 2011.
[66] E. Meijer, B. Beckman, and G. M. Bierman. LINQ: Reconciling objects, relations, and XML in the .NET framework. In *International Conference on Management of Data (SIGMOD)*, pages 706–706, 2006.
[67] M. Mendell, H. Nasgaard, E. Bouillet, M. Hirzel, and B. Gedik. Extending a general-purpose streaming system for XML. In *Conference on Extending Database Technology (EDBT)*, pages 534–539, 2012.
[68] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks. Incremental update of datalog materialisation: the backward/forward algorithm. In *Conference on Artificial Intelligence (AAAI)*, pages 1560–1568, 2015.
[69] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A timely dataflow system. In *Symposium on Operating Systems Principles (SOSP)*, pages 439–455, Nov. 2013.
[70] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.
[71] F. Peng and S. S. Chawathe. XPath queries on streaming data. In *International Conference on Management of Data (SIGMOD)*, pages 431–442, 2003.
[72] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal of Data Semantics*, 10:133–173, 2008.
[73] M. Pouzet. *Lucid synchrone, version 3, Tutorial and reference manual*, 2006.
[74] Y. Ren and J. Z. Pan. Optimising ontology stream reasoning with truth maintenance system. In *Conference on Information and Knowledge Management (CIKM)*, pages 831–836, 2011.
[75] A. V. Riabov, E. Bouillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan. Wishful search: Interactive composition of data mashups. In *International Conference on World Wide Web (WWW)*, pages 775–784, 2008.
[76] N. Seyfer, R. Tibbetts, and N. Mishkin. Capture fields: Modularity in a stream-relational event processing language. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 15–22, 2011.
[77] R. Soulé, M. Hirzel, B. Gedik, and R. Grimm. River: An intermediate language for stream processing. *Software – Practice and Experience*, 46(7):891–929, July 2016.
[78] R. Soulé, M. Hirzel, R. Grimm, B. Gedik, H. Andrade, V. Kumar, and K.-L. Wu. A universal calculus for stream processing languages. In *European Symposium on Programming (ESOP)*, pages 507–528, 2010.
[79] R. Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.
[80] W. Thies, M. Karczmarek, M. Gordon, D. Maze, J. Wong, H. Hoffmann, M. Brown, and S. Amarasinghe. StreamIt: A compiler for streaming applications. Technical Report LCS-TM-622, MIT, 2002.
[81] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm @Twitter. In *International Conference on Management of Data (SIGMOD)*, pages 147–156, June 2014.
[82] E. D. Valle, D. Dell'Aglio, and A. Margara. Taming velocity and variety simultaneously in big data with stream reasoning. In *Conference on Distributed Event-Based Systems (DEBS) Tutorial*, pages 394–401, 2016.
[83] A. van Wijngaarden, B. Mailloux, J. Peck, C. Koster, M. Sintzoff, C. Lindsey, L. Meertens, and R. Fisker. *Revised Report on the Algorithmic Language ALGOL 68*. 1975.
[84] M. Vaziri, O. Tardieu, R. Rabbah, P. Suter, and M. Hirzel. Stream processing with a spreadsheet. In *European Conference on Object-Oriented Programming (ECOOP)*, pages 360–384, 2014.
[85] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Complex event processing over uncertain data. In *Conference on Distributed Event-Based Systems (DEBS)*, pages 253–264, 2008.
[86] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *International Conference on Management of Data (SIGMOD)*, pages 407–418, 2006.
[87] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Symposium on Operating Systems Principles (SOSP)*, pages 423–438, Nov. 2013.
[88] F. Zemke, A. Witkowski, M. Cherniak, and L. Colby. Pattern matching in sequences of rows. Technical report, ANSI Standard Proposal, 2007.
[89] Q. Zou, H. Wang, R. Soulé, M. Hirzel, H. Andrade, B. Gedik, and K.-L. Wu. From a stream of relational queries to distributed stream processing. In *Conference on Very Large Data Bases (VLDB) Industrial Track*, pages 1394–1405, 2010.

Kenneth Ross Speaks Out on Making Contributions that Span Technology and Science

Marianne Winslett and Vanessa Braganholo

Kenneth Ross

http://www.cs.columbia.edu/~kar/

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Snowbird, Utah, USA, site of the 2014 SIGMOD and PODS conference. I have here with me Ken Ross, who is a professor at Columbia University. He was a Sloan Fellow and a Packard Fellow, as well as an NSF Young Investigator. His Ph.D. is from Stanford.

So, Ken, welcome!

What is next for main memory databases?

So, I think as we go forward, main memory databases are going to be the mainstream databases. We're going to be thinking of disk-resident databases as secondary. The primary copy of the data and the primary activity is going to be in main memory. IO and so on are going to be things you think about just for recovery and persistence. So main memory databases are the mainstream. The obvious questions are how to get very fast transaction processing, very fast analytics. As my research interests kind of reflect, the hardware is evolving relatively rapidly – you're getting multi-core machines, and on these multi-core machines, you're getting various kinds of hardware capabilities. I think the critical thing going forward is making the best use of these hardware capabilities. Things like SIMD units, things like transactional memory, gather instructions, relatively low-level things but they can make a very significant performance difference when they're in the inner loop of a database join, or aggregation or some important operation that is run many billions of times.

I think the Database field is particularly good, perhaps better than the many other fields in and out of Computer Science, in admitting work that goes all the way from theory to systems.

I can see how that would be really important, but isn't the rate of data collection expanding faster than memory size?

The rate of data collection is growing very rapidly. People are trying to scale systems accordingly. The RAM sizes you see are dramatically increasing. That being said, some of the biggest customers of Oracle, for example, still have datasets that reside in large main memories. Oracle is actually supplying main memory systems to those customers. So, while there will always be applications on the fringe that are in excess of what we can store in main memory, things like large astronomy datasets and so on, many of the applications, a lot of the critical ones for economic or scientific analysis will have working sets that do fit in main memory and can benefit from these sorts of technologies.

For our younger readers, can you say a few words about how being in main memory changes everything compared to the old disk-based days?

Okay, so in the disk-based days, you would have to wait for an IO to do most operations. An IO would take a few milliseconds for the data to come in and if you had to read that disk page to find out what the next item you need to read is, then you have to wait for a second IO that would have to be in sequence, you can't make them concurrent. So, you have a lot of latency. Once the data is in main memory, you are now thinking about nanoseconds rather than milliseconds, so you have six orders of magnitude potential difference in speed to access data. Then you start caring not about whether it's on disk or in memory but is it in memory or is it in the cache? The caches can be a factor of 50 to 100 faster than the main memory in terms of access. You have similar problems in terms of trying (from a disk-based database) to buffer the disk-resident data in RAM – you see analogous problems at high levels of the memory hierarchy trying to put the data you need not in RAM, but in the cache (at least for a short period), to take advantage of temporal locality and get faster performance.

We're still teaching them in the courses about B-trees. Everything is based on B-trees. Are we teaching them the right thing?

In my database implementation class, we start out teaching them about B-trees, so I think for historical reasons, it's important to be founded in that kind of knowledge, but then we go on and talk about cache sensitive B+ trees[1] which is actually something I worked on with my student Jun Rao back in 2000. You take the basic B-tree structure and you re-work it to make it work well in main memory. So instead of having k+1 pointers and k data items, you'll have 2k data items and 1 pointer. So, you can fit much more in a node, and you size the node to be the size of a cache line. You get away with this because the pointer now points to a contiguous set of child nodes and so as a result, you can use arithmetic to figure out where the child node is. Also, you get a much higher branching factor for one cache line, so you get fewer cache misses during a

[1] Jun Rao, Kenneth A. Ross: Making B+-Trees Cache Conscious in Main Memory. SIGMOD Conference 2000: 475-486.

traversal. So, the underlying concept of a B-tree is still there, but as a research community, we have evolved it to suit the appropriate kinds of memory technologies that are appropriate for the time. So, I would hope, and I do get told by some of my colleagues at other institutions that things like cache sensitive B+ trees are now being taught in the mainstream database classes.

Excellent. Is your research valuable to industry?

I think it's very valuable to industry. I have some collaboration with companies like Oracle where they're taking interns from my group. Some of the problems that we're working on, for example, there is a paper[2] that we have at SIGMOD 2014 on track joins that is motivated by a problem at Oracle where you have very large joins over a cluster of network machines. My student Orestis and I have come up with a technique to do joins kind of like a semi-join where on a key-by-key basis, the data is re-partitioned which ends up transferring a lot less data than say if you did hashing. This is a critical workload for Oracle. This is the slowest query in an important customer workload that we were able to speed up significantly. So that is just one example of where these research techniques can have a fairly direct impact.

To have that impact you had to know that it was the slowest query for an important customer. So how do you build that relationship where you learn those types of things?

I think in this particular example, Orestis, my student, was the key player. He was there as an intern. He found out about what was going on in Oracle. I don't take all the credit, but I perhaps can take credit for placing Orestis in Oracle and working with Eric Sedlar, for example, to find a project where Orestis's strengths could be most utilized. It's been fruitful, and we have an ongoing collaboration with Oracle.

Your Ph.D. work was entirely theoretical, but most of your subsequent research was on the system side. Do you have tips for making that transition?

It was a complex thought process I had to go through as I joined Columbia as a junior faculty member. I did a thesis in which I developed the well-founded semantics for Datalog[3,4] and that was actually relatively influential. It had a number of citations and formed the basis for a lot of work in Datalog. So, I was sort of branded as being the person, along with my collaborators John Schlipf and Allen Van Gelder, who developed the well-founded semantics. And this is a blessing and a curse in a way because as I progressed in my career at Columbia, Datalog kind of went out of fashion. If you're working in a field that kind of goes out of fashion and people point at you and say, "Oh he did the important Datalog thing," even if you are doing other things, that's what they remember you for. So, it takes some effort to actually take what you're doing next and make it known in the community. In the period before I was coming up for tenure for example, I went to various other institutions and labs and gave talks about some of the work I was doing on query processing and optimization and so on which was more applied and it was sort of hitting much more in the direction in which I find myself right now.

Do you think that a young person starting their first tenure-track job today could make that big switch to systems where you had to really build things before you can publish – would they have time to make such a big transition before tenure?

That's a tricky question. I think the best students are able to span theoretical and practical concerns. I think the Database field is particularly good, perhaps better than the many other fields in and out of Computer Science, in admitting work that goes all the way from theory to systems. Even though I have moved from theory to systems over the years, it just means I used to publish in PODS, and now I publish in SIGMOD and VLDB more, but it's still the same conferences, I still circulate with the same people, and I think that interaction is good.

So, coming back to your question, I think there are theoretical people who prefer to work on purely theoretical problems, and that's fine, but if you're a theoretical person who has an inclination to write code and implement things like I like to do, I think it's fun to play with that. Don't necessarily invest all of your energy in that. Have one or perhaps two side projects that may or may not pan out. Keep your mainstream work that you feel you have the most cutting-edge advantage in your research going, but do these side

[2] Orestis Polychroniou, Rajkumar Sen, Kenneth A. Ross: Track join: distributed joins with minimal network traffic. SIGMOD Conference 2014: 1483-1494.

[3] Allen Van Gelder, Kenneth A. Ross, John S. Schlipf: Unfounded Sets and Well-Founded Semantics for General Logic Programs. PODS 1988: 221-230.

[4] Allen Van Gelder, Kenneth A. Ross, John S. Schlipf: The Well-Founded Semantics for General Logic Programs. J. ACM 38(3): 620-650 (1991).

projects. These side projects can often expand and become products that take a life of their own, and they can drive you in these new directions. That was kind of what happened for me.

In some sense, you've come full circle because your recent sabbatical at LogicBlox involved working with Datalog. Has the time finally come for Datalog in our field?

That's a very interesting question. I did get approached by the principal people at LogicBlox, including Molham Aref, and they sort of looked at me as if I was this really famous rock star type character because I had done well-founded semantics. And here I am 20 years later, having put that in my past and not being used to people thinking of me in that way. I have to say it was flattering. I enjoyed the attention from having that kind of feedback. Then it led to some interactions, and as a result, I did go to LogicBlox and did some work while I was there related to some of the interesting problems they were having, that overlapped with the research I had done in the past. So that kind of recapitulated and I looked at it in a new way that might be relevant to LogicBlox. And even after my sabbatical, I've managed to continue having a consulting relationship to LogicBlox that I think is helpful for both sides.

Coming back to your question about if this is the time for Datalog… what I really like about Datalog is its declarativeness. I think that SQL has succeeded in the relational database community because it's declarative. People don't know how to program yet they can write SQL queries, so it takes less effort, energy, and knowledge to master that technology. Datalog has the disadvantage that it is a logic language and people are often not as inclined to think in a logical framework in terms of predicate calculus and so on. On the other hand, sometimes you can use syntactic sugar to hide some of those complexities. With Datalog, you can use recursion to express things declaratively. Some of the work for example, by Joe Hellerstein in Berkeley is using recursion to reason about time and protocols and so on. I think that's an excellent kind of direction because it's taking advantage of the declarativeness, but using a fairly limited expressive power language to write your specifications so that you can reason about them, prove correctness results, and form a layer of abstraction that is much cleaner than an arbitrary procedural code. So, I think it's cool that these additional applications that weren't really foreseen for Datalog have come along and are making it relevant again.

You are unusual among computer scientists in bringing a broad scientific perspective to your work. In fact, most people don't know that you've co-authored published articles in physics.

Yes, when I was an undergraduate student, as a summer project I worked with an applied mathematician who worked in physics on a model of the Ising spin chain. I don't actually understand it in full detail, but I did some coding of some physical simulations that corresponded to the physical problem that he was studying. It turned out that the results were kind of interesting. They showed a fractal structure that was somewhat new, and as a result, we got a couple of publications in theoretical physics journals.

I like the idea of working in the scientific field itself, trying to understand the domain rather than just building a tool to help the domain scientists and I think that provides a much more satisfying and rounded type of experience in making a contribution.

And you're still working in science, although more recently it's been bio-informatics. How is it to work with bio people?

Some of this interest in biology came from a point where the human genome was about to be sequenced. The various universities were being called to help work in the sequencing of the human genome, and some people from the Medical School at Columbia came down to the Engineering School to try to recruit people to work on the sequencing effort. In order to get them interested, they gave a little short course on biology, a five-lecture sequence in which they taught basic biology to engineers. So, I attended this course, and I was actually fascinated by some of the biology. I didn't get involved in the sequencing effort at the time, but it got me thinking about many of these questions of biology.

Over the years, I've actually worked on a couple of research questions in biology sort of on my own as one of these pet projects as I've mentioned before. For

example, you might not know I wrote an article[5] about why some groups of species have a very variable chromosome number among different species in a clade while other groups of species have a very conserved number of chromosomes within a clade.

More recently, I've been thinking about autoimmune diseases, and I have an article studying the genetics of autoimmune disease with the hypothesis that the cause of autoimmunity is an immune response against mutated genes (mutated proteins that are expressed in the body). In order to explore this, I took the human referenced genome and ran some SQL queries on these referenced genomes from the UCSC database[6] and found a statistically significant overrepresentation of genes with very long repeat regions among auto antigens. This was kind of exciting, and I wrote it up. It required a lot of reading and a lot of understanding of the biological literature. It appeared at PLoS One[7].

That work I did on my own, but in the biology field, people don't really take you seriously until you've got an experimental validation of your ideas. I talked with one of my colleagues who does computational biology, and he recommended I speak to a certain person who studies inflammatory valve disease at the Mount Sinai Hospital. Her name is Judy Cho and so I'm working with her and some other people at Mount Sinai Hospital to experimentally validate this hypothesis. So, in this particular case, I've worked somewhat on my own but then done the collaboration afterward.

Coming up for tenure, it's important to have your work known by the community. Give talks about your best work and visit other labs.

I like the idea of working in the scientific field itself, trying to understand the domain rather than just building a tool to help the domain scientists and I think that provides a much more satisfying and rounded type of experience in making a contribution. Some of this work was inspired by my Packard Foundation Fellowship. You did mention I was a Packard Fellow (from 1993-1998) and one of the things that the Packard Fellowship does is that it brings all of the Packard fellows together and they give talks about their work. Just as an aside, one of my fondest memories was getting a pat on the back from David Packard as I gave my talk at the Packard Fellows meeting. And so, I'm up there giving my talk, at the time I was doing some theoretical work on object-oriented databases, and these other scientists were talking about cures for Malaria and various other very high impact things. I was kind of scratching my head thinking, "Here I'm doing object-oriented database theory, and these people are really impacting the world." I kind of had this urge; I want to impact the world too. I stayed in my main area of Computer Science, and I still worked on databases, but I had a strong incentive to do one or two of these pet projects to try to explore things outside that domain. I just followed my curiosity and had fun, so that's how this biology project eventuated.

It sounds amazing! Stepping back for a moment to the validation, is that going to be more SQL queries over particular patients' genome or is this stuff they're going to do in a wet lab?

These will be wet lab experiments. There's a particular technology that allows you to sequence genomes in particular regions with fairly long reads. That will enable you to look for certain structures that should, according to the hypothesis, differ between patients and controls. These structures are actually not easy to detect with current technologies because they're longer than the read length that most of these short read technologies give. So, the nice thing about this collaboration at Mount Sinai Hospital is that they have this database of 30,000 patient's blood samples that they can go to, and you can get 100 people with a certain disease and get their blood samples and test them versus controls at relatively low overhead. It takes some effort to setup the scientific experiment, and there are all kinds of design issues for the experiment that are things that I wouldn't have thought of at first, but my collaborators there have to go through to make sure that the experiment is going to succeed and find the things we're looking for. That's where it's essential to have collaborations because I have no wet lab experience and we need to bring out our respective strengths to be able to solve these bigger problems.

By thinking like a computer scientist, you came up with this hypothesis for autoimmune disease. Does that mean you have a hypothesis for how to cure them?

If this is, in fact, the mechanism that causes autoimmune disease, which is speculative at this point because it's just a statistical association, it's not validated, so I do

[5] Kenneth A. Ross: Alpha radiation is a major germ-line mutagen over evolutionary timescales. Evolutionary Ecology Research, 2006, 8: 1013–1028.

[6] https://genome.ucsc.edu/

[7] Kenneth A. Ross: Coherent Somatic Mutation in Autoimmune Disease. PLoS One. 2014 Jul 2; 9(7): e101093.

not want to claim that this is the solution to autoimmune disease. But let's imagine for a moment that in fact, somatic mutation of these proteins is what triggers autoimmune disease. It opens up certain possibilities. If you know the specific proteins that might be causing the autoimmune disease that have been mutated in a way that is relatively deterministic and predictable, you could do various things relative to that particular protein. You can try to induce tolerance to that protein, for example, or you could find ways to take that protein out of circulation one way or another. My knowledge of biology is limited in terms of knowing the different options for which you might use that knowledge, but if you know the basic procedures and steps that trigger a disease, you can go early in this triggering process, identify the early players and try to get things as close to the causative part of the mechanism as possible. So, by extending the knowledge base and by making the knowledge closer to the triggering point and making the identification of very specific targets, I think that opens up much more opportunity compared to alternatives like just generally dampening the immune system, which can be effective and is the current treatment for many autoimmune diseases but is non-specific to the particular causative factor.

Do you have any words of advice for fledging or mid-career database researchers?

So, for fledging database researchers, I wouldn't worry too much in the first year or two about having lots of publications and so on. It takes a while to get started. So, settle down, maybe write a grant proposal, get comfortable with teaching, find students, don't set high expectations about publishing two big papers a year during those first couple of years. Be easy on yourself as you ramp up.

By mid-career are you suggesting before or after tenure?

Either way.

Ok, so leading up to tenure, I think it's important to focus on the tenure process. One of the nice things about having tenure is that you can choose these arbitrary pet projects and even choose to spend most of your time on those and you have the academic license to do so. Before tenure, there is maybe a little bit of a risk if you spend a majority of the time on those because if they don't pan out, you would not have enough to show. So, maybe limit yourself to one pet project before tenure and maybe branch out afterward. Coming up for tenure, it's important to have your work known by the community. Give talks about your best work and visit other labs. The crucial thing about the tenure process, having seen it from both sides, is the quality of the letters of recommendation. You want to get letters from people who know about the impact of your work. So, tell people about the impact of your work at conferences. Do the circulating among the people in the field, particularly the senior people. Give them the elevator pitch if necessary or try to sit down with them for longer periods and communicate your work to get it as well-known as possible.

The crucial thing about the tenure process, having seen it from both sides, is the quality of the letters of recommendation. You want to get letters from people who know about the impact of your work.

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

I guess I have several favorites. I like the well-founded semantics I did in my Ph.D. thesis because it had high impact. Even now, it has many citations[8] and it sort of resolved a question that many people had posed for a while. So that was satisfying, and I enjoyed that work for that reason. Some other work that I like, I particularly like the cache conscious B+ tree work that I referred to earlier. I think we got in pretty early. I don't think many people at that time appreciated how important the cache was in the database community. I think we were trendsetters in that regard and this particular paper has influenced how people design indexes now and now it's regularly routine to make indexes cache sensitive or cache aware in various ways. Maybe it's too early to think of this biology paper that I published at PLoS One as my favorite, but this was sort of a major undertaking, it was a lot of fun doing and a lot of work, and it's something brand new. I have a fond feeling about it.

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

[8] The PODS paper has 591 citations, and the JACM paper has 2042 citations.

Okay, so I imagine many people would say they'd like to do more coding and that as a faculty member you would distribute the coding tasks to your students and not code yourself. I actually do find that I get time to code and I like coding and programming and so on. That is something I already do. I think the thing that I would like to do beyond that is to explore new domains. For example, in this biology application, I'm reliant on these other people doing the wet lab experiments and do the sequencing and so on and I know I have some colleagues at Columbia who started out as geneticists doing the theoretical work and basically evolved overtime and took courses to master the wet lab work and so on. I think it would be fun to do that sort of thing, to try to learn the technologies dealing with biological reagents and so on. I think that would take a fairly big investment of time and I'm not sure I have the time to do that, but if I had spare time, I think it would be fun to get to that point that I could be competent at doing those things and eventually direct others to do wet lab experiments in support of these biological hypotheses.

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

Okay, so there was the big biological revolution when the genome was sequenced where everyone was looking at these questions, sequencing and so on. At the time when that happened, I actually questioned whether I had chosen the right field. I thought to myself, okay if I had been doing my Ph.D. ten years later, might I have chosen to go into genetics or bioinformatics or something like that instead of computer science? In retrospect, I think I'm in a good position now because having done database work and being able to pick up a lot of this biology, I can make contributions that kind of span the technology and the science. A lot of the stuff that came out early in the genome revolution was technology that became obsolete over time. Things change a lot, and if you end up investing too much in a particular technology and with time that goes obsolete, then that is not so useful knowledge. That was hard to see at the beginning of that time. So, I think maybe if I were to change something, it would be to learn more biology sooner, to be able to work on these problems, but I think that at least in the subproblems that I've been working on, I've been able to catch up so to speak.

[…] having done database work and being able to pick up a lot of this biology, I can make contributions that kind of span the technology and the science.

Thank you very much for talking with me today.

You are welcome.

Paris Koutris Speaks Out on a Theoretical Model for Query Processing

Marianne Winslett and Vanessa Braganholo

Paris Koutris
http://pages.cs.wisc.edu/~paris/

Welcome to the ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we're at SIGMOD 2017 in Chicago. I have with me Paris Koutris, who won the 2016 ACM SIGMOD Jim Gray Dissertation Award for his thesis entitled "Query Processing in Massively Parallel Systems." Paris is now a professor at the University of Wisconsin-Madison, and he did his Ph.D. work with Dan Suciu at the University of Washington.

So, Paris, welcome!

It is great to be here.

What is your thesis about?

My thesis has to do with query processing in massively parallel systems. The key observation is that due to the data explosion we've seen over the last years, there's a massive volume of data being around, and we have to process this data. In order to process this data fast, one way is to use parallelism. This has led to an explosion of different types of systems – distributed systems, parallel systems – that try to improve performance. My work has to do with how we can theoretically model these types of systems and how we can formally reason about these systems.

My first contribution was introducing a model which we called the Massively Parallel Computation model (or MPC for short) that basically creates a theoretical framework to analyze query processing. This model has two main parameters. The first one is communication, so it measures how much data is being exchanged, and the other is the number of rounds or synchronizations. This measures how often does the system have to synchronize and wait for all the machines to reach the same point before moving forward. Using this model, my thesis analyzed different types of algorithms for join processing. Joins are the backbone of any database system. And so, what we did is try to find out if there is a tradeoff between communication and synchronization, and how can we model it, and not only try to create new algorithms, but also try to give lower bounds on how well these algorithms can perform. This is the main part of my thesis.

The second part has to do with what we can do further. For example, many times, data has skew, which means that there are some values in the data that appear more often than the others, and that can create an imbalance in query processing. In this case, we have to use different types of techniques to deal with skew. And my thesis also tried to reason about these types of problems.

Does that mean that you introduced new join algorithms yourself or improved the existing ones?

It's actually both. In the thesis, we both analyzed existing algorithms and proved new bounds on how well they can do, and also introduced some techniques that were novel and could be actually used in practice.

What kind of new join technique did you use?

For example, the new technique is how we can deal with skew. Typically, when we are doing a hash join algorithm, we are distributing the elements by hashing a particular value. Now if this value appears very often, there will be skew, so there's going to be a struggler in a machine that will end up doing more work than the other machines. In order to deal with this problem, we essentially have to find out which are the values that have skew and split up their work in more machines. And we have to do it in a very particular way so that we can get the best possible performance.

Given that this is a classic issue, I find it very surprising that people hadn't already come up with techniques to do a better job of spreading key values.

There are existing techniques to do that, but what we did is we showed which are the theoretical optimal techniques that you can use. So, for some cases, we did use some existing tools. For some others, we had to introduce new ways of balancing that were theoretically optimal.

How close are we now to the theoretical optimal lower bound?

This is an excellent question. In some cases, some of these new theoretical ideas have proved to be faster in practice than the typical algorithms. But there are cases where the constants in the theoretical analysis are so large that going back to some of the classic techniques is faster.

The issue here – and this is generally an issue with theoretical analysis – is that we make worst-case assumptions about the data: for example, we are assuming that we're analyzing the worst case that can happen. And of course, for many real-world datasets this is far away from the truth. So, a very exciting direction is to try to incorporate this assumption in the analysis and try to see how you can prove that the analysis of an algorithm theoretically matches the behavior that we see in the real systems.

What was your model like? Is it based on queuing theory or another approach?

The model is actually very close to the BSP model by Valiant, the Bulk Synchronous Model[1]. The idea is that processing operates in rounds, and at each round there is some communication, some computation, and then

[1] Leslie G. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.

there is a synchronization barrier. But in order to make our analysis feasible, we abstracted away some of the parameters of Valiant's model. For example, we ignore the computation and try to figure out how well the data is balanced across the different machines that we have.

That topic sounds quite classical and not very Dan Suciu like. Where did the topic come from?

Excellent question. The story is interesting. When I started my Ph.D. at the University of Washington, I started talking with Dan on possible projects I could do. And he was talking about probabilistic databases and all the other things he has been doing, and then he also mentioned this idea of "Oh, people like doing parallel join processing, and we don't know yet how to analyze this", and I got immediately attracted to that problem and started working on that. And I think that turned out very well. So, it was kind of by luck that I started working on this project, but it was very interesting.

[…] by talking with more people and collaborating with more people, you're going to come across with many different ideas. And that may actually improve your research.

Do you have any words of advice for graduate students?

Yes. One thing that I think is very important, and sometimes in this competitive environment where you are trying to publish as many papers as possible it is kind of lost, is not only to do research but to also try to talk with as many people as possible and try to network with as many people as possible. And also try to develop collaborations with many people – other students in your department, possibly other professors in your department, or also other people and other students from other departments. And the way I view it is that, if you plan to stay in academia, these people will actually be your colleagues for the rest of your career. That's one thing. And second, by talking with more people and collaborating with more people, you're going to come across with many different ideas. And that may actually improve your research.

So, the second thing that I want to say is that students should not be afraid to tackle new problems. It's probably easier to look at some existing papers and then try to improve upon these or try to think about a new technique that gets an improvement of 10 percent in the performance. But I think it's much more impactful if you try to go to new areas and try to introduce new problems, new frameworks, and in general, try to explore new things. The disadvantage of that is that it will be harder, possibly, to convince the database community that this is an important problem, and we need to do research on that. But on the other hand, the results – the potential of this type of research is much higher.

Did you have trouble convincing the community that your particular topic was something they should care about?

I would say no for my case, but I've come across many other cases where this has happened. So, I know that this is an issue and a danger if you're trying to do these types of things. And my point is that you should not be discouraged if this happens, and you should try to push through these directions.

Alright. Well, thank you very much for talking with us today.

Yeah, it was very nice being here. Thank you very much

The New and Improved SQL:2016 Standard

Jan Michels
Oracle Corporation
jan.michels@oracle.com

Keith Hare
JCC Consulting
keith@jcc.com

Krishna Kulkarni
SQL Consultant
krishna7277@gmail.com

Calisto Zuzarte
IBM Corporation
calisto@ca.ibm.com

Zhen Hua Liu
Oracle Corporation
zhen.liu@oracle.com

Beda Hammerschmidt
Oracle Corporation
beda.Hammerschmidt@oracle.com

Fred Zemke
Oracle Corporation
fred.zemke@oracle.com

ABSTRACT

SQL:2016 (officially called ISO/IEC 9075:2016, *Information technology — Database languages — SQL*) was published in December of 2016, replacing SQL:2011 as the most recent revision of the SQL standard. This paper gives an overview of the most important new features in SQL:2016.

1. INTRODUCTION

The database query language SQL has been around for more than 30 years. SQL was first standardized in 1986 by ANSI as a US standard and a year later by ISO as an international standard, with the first implementations of SQL preceding the standard by a few years. Ever since first published, the SQL standard has been a tremendous success. This is evidenced not only by the many relational database systems (RDBMSs) that implement SQL as their primary query[1] language but also by the so-called "NoSQL" databases that more and more see the requirement (and value) to add an SQL interface to their systems. One of the success factors of SQL and the standard is that it evolves as new requirements and technologies emerge. Be it the procedural [23], active database, or object-relational extensions that were added in the 1990s [22], the XML capabilities in the 2000s [19], temporal tables in the early 2010s [17], or the many other features described in previous papers [18], [20], [21], and not the least the features described in this paper, the SQL standard has always kept up with the latest trends in the database world.

SQL:2016 consists of nine parts [1]-[9], all of which were published together. However, with the exception of Part 2, "Foundation" [2], the other parts did not significantly change from their previous versions, containing mostly bug fixes and changes required to align with the new functionality in Part 2. As with the previous revisions, SQL:2016 is available for purchase from the ANSI[2] and ISO[3] web stores.

A high-level theme in SQL:2016 is expanding the SQL language to support new data storage and retrieval paradigms that are emerging from the NoSQL and Big Data worlds. The major new features in SQL:2016 are:

- Support for Java Script Object Notation (JSON) data
- Polymorphic Table Functions
- Row Pattern Recognition

SQL:2016 also includes a number of smaller features, such as additional built-in functions.

In addition to the formal SQL standard, the SQL committee has developed a series of Technical Reports (TRs). TRs, while non-normative, contain information that is useful for understanding how the SQL standard works. The SQL Technical Report series contains seven TRs [10] – [16] that are available at no charge from the ISO/IEC JTC1 "Freely available standards" web page[4].

The remainder of this paper is structured as follows: section 2 discusses the support for JSON data, section 3 discusses polymorphic table functions, section 4 discusses the row pattern recognition functionality, and section 5 showcases a select few of the smaller enhancements.

2. SUPPORT FOR JSON DATA

JSON [24] is a simple, semi-structured data format that is popular in developer communities because it is well suited as a data serialization format for data interchange. JSON data is annotated and the format allows nesting making it easy-to-read by both humans and machines. Many database applications that would benefit from JSON also need to access "traditional" tabular data. Thus, there is great value in storing, querying, and manipulating of JSON data inside an RDBMS as well as providing bi-directional conversion between relational data and JSON data.

To minimize the overhead of a new SQL data type, SQL:2016 uses the existing SQL string types (*i.e.*, either character strings like VARCHAR and CLOB, or

[1] "Query" in this context is not restricted to retrieval-only operations but also includes, among others, DML and DDL statements.

[2] http://webstore.ansi.org/

[3] https://www.iso.org/store.html

[4] http://standards.iso.org/ittf/PubliclyAvailableStandards/

binary strings like BLOB) to carry JSON values. Since there is no standard JSON query language yet, the SQL standard defines a path language for navigation within JSON data and a set of SQL built-in functions and predicates for querying within a JSON document.

We will illustrate the SQL/JSON[5] path language and SQL/JSON operators in the following sections. Due to space restrictions, we cannot cover all SQL/JSON features. For a detailed description of the SQL/JSON functionality the interested reader is referred to [15].

2.1 Querying JSON in SQL

2.1.1 Sample data

Our examples will use the table `T` shown below:

ID	JCOL
111	{ "Name" : "John Smith", "address" : { "streetAddress": "21 2nd Street", "city": "New York", "state" : "NY", "postalCode" : 10021 }, "phoneNumber" : [{ "type" : "home", "number" : "212 555-1234" }, { "type" : "fax", "number" : "646 555-4567" }] }
222	{ "Name" : "Peter Walker", "address" : { "streetAddress": "111 Main Street", "city": "San Jose", "state" : "CA", "postalCode" : 95111 }, "phoneNumber" : [{ "type" : "home", "number" : "408 555-9876" }, { "type" : "office", "number" : "650 555-2468" }] }
333	{ "Name" : "James Lee" }

In `T`, the column `JCOL` contains JSON data stored in a character string.

Curly braces { } enclose JSON objects. A JSON object has zero or more comma-separated key/value pairs, called members. The key is a character string before a colon; the value is a JSON value placed after a colon. For example, in each row, the outermost JSON object has a key called "Name" with varying values in each row.

Square brackets [ ] enclose JSON arrays. A JSON array is an ordered, comma-separated list of JSON values. In the first and second rows, the key called "phoneNumber" has a value which is a JSON array. This illustrates how JSON objects and arrays can be nested arbitrarily.

Scalar JSON values are character strings, numbers, and the literals `true`, `false` and `null`.

The sample data is fairly homogeneous, but this is not a requirement of JSON. For example, the elements of an array do not need to be of the same type, and objects in different rows do not have to have the same keys.

2.1.2 IS JSON predicate

The `IS JSON` predicate is used to verify that an SQL value contains a syntactically correct JSON value. For example, this predicate can be used in a column check constraint, like this:

```
CREATE TABLE T (
  Id INTEGER PRIMARY KEY,
  Jcol CHARACTER VARYING ( 5000 )
           CHECK ( Jcol IS JSON ) )
```

The preceding might have been used to create the table `T`, insuring that the value of `Jcol` is valid JSON in all rows of `T`.

In the absence of such a constraint, one could use `IS JSON` as a filter to locate valid JSON data, like this:

```
SELECT * FROM T WHERE Jcol IS JSON
```

2.1.3 SQL/JSON path expressions

The remaining SQL/JSON operators to query JSON use the SQL/JSON path language. It is used to navigate within a JSON value to its components. It is similar to XPath for XML and also somewhat similar to object/array navigation in the JavaScript language. A path expression is composed of a sequence of path steps; each step can be associated with a set of predicates.

2.1.4 JSON_EXISTS predicate

`JSON_EXISTS` is used to determine if an SQL/JSON path expression has any matches in a JSON document. For example, this query finds the IDs of the rows with a key called “address”:

```
SELECT Id
FROM T
WHERE JSON_EXISTS ( Jcol,
        'strict $.address' )
```

The example works as follows. The first argument to `JSON_EXISTS`, `Jcol`, specifies the context item (JSON value) on which `JSON_EXISTS` operates. The keyword `strict` selects the strict mode; the

[5] The support for JSON is specified in Foundation [2] and not in a separate part (as is done, *e.g.*, for SQL/XML [9]). Still, the moniker SQL/JSON is associated with the JSON-specific functionality in SQL.

alternative is `lax`. As its name implies, strict mode expects that the JSON document conforms strictly to the path expression, whereas lax mode relaxes some of these expectations, as will be seen in later examples. `$.address` is the path expression that is applied to the context item. In the path expression, `$` is a variable referencing the context item, the period is an operator used to navigate to a key/value pair within a JSON object, and `address` is the name of the desired key. The `JSON_EXISTS` predicate will be true if this path expression successfully finds one or more such key/value pairs. With the sample data, the query will find the IDs 111 and 222 but not 333.

2.1.5 JSON_VALUE function

The `JSON_VALUE` function is used to extract a scalar value from a given JSON value. For example, to find the value of the "Name" key/value pair in each row, one could use this query:

```
SELECT JSON_VALUE ( Jcol,
   'lax $.Name' ) AS Name
FROM T
```

This example uses lax mode, which is more forgiving than strict mode. For example, it is common to use a singleton JSON value interchangeably with an array of length one. To accommodate that convention, in lax mode, if a path step requires an array but does not find one, the data is implicitly wrapped in a JSON array. Conversely, if a path step expects a non-array but encounters an array, the array is unwrapped into a sequence of items, and the path step operates on each item in the sequence.

The following query might be used to find the first phone number in each row.

```
SELECT Id, JSON_VALUE ( Jcol,
   'lax $.phoneNumber[0].number' )
   AS Firstphone
FROM T
```

JSON arrays are 0-relative, so the first element is addressed `[0]`. The last row of sample data has no such data; in that case, lax mode produces an empty sequence (instead of an error) and `JSON_VALUE` will return a null value. The result of the query is:

ID	FIRSTPHONE
111	212 555-1234
222	408 555-9876
333	

In the last row above, the `FIRSTPHONE` cell is blank, indicating an SQL null value, a convention we will use throughout this paper.

Or suppose the task is to find all fax phone numbers. The query to solve this is

```
SELECT Id, JSON_VALUE ( Jcol,
   'lax $.phoneNumber
      ? ( @.type == "fax" ).number' )
    AS Fax
FROM T
```

This query illustrates a filter, introduced by a question mark and enclosed within parentheses. The filter is processed as follows: since the query is in lax mode, the array `$.phoneNumber` is unwrapped into a sequence of items. Each item is tested against the predicate within the parentheses. In this predicate, the at-sign `@` is a variable bound to the item being tested. The predicate `@.type == "fax"` is true if the value of the `"type"` member equals `"fax"`. The result of the filter is the sequence of just those items that satisfied the predicate. Finally, the member accessor `.number` is applied, to obtain the value of the member whose key is `"number"`. The result of the query is:

ID	FAX
111	646 555-4567
222	
333	

All of these examples returned character string data, the default return type of `JSON_VALUE`. Optional syntax can be used to specify other return types, as well as various options to handle empty or error results.

2.1.6 JSON_QUERY function

`JSON_VALUE` can only extract scalars from a JSON value. The `JSON_QUERY` function, on the other hand, is used to extract a fragment (*i.e.*, an SQL/JSON object, array, or scalar, possibly wrapped in an SQL/JSON array, if the user specifies this) from a given JSON value. For example, to obtain the complete value of the "address" key, this query might be used:

```
SELECT Id, JSON_QUERY ( Jcol,
   'lax $.address' ) AS Address
FROM T
```

With the following results[6]:

[6] The result shows some insignificant pretty-printing whitespace. The SQL standard does not prescribe this. A conforming implementation is allowed to either add or omit insignificant whitespace. Here it is only shown for readability.

ID	ADDRESS
111	{ "streetAddress": "21 2nd Street", "city": "New York", "state" : "NY", "postalCode" : 10021 }
222	{ "streetAddress": "111 Main Street", "city": "San Jose", "state" : "CA", "postalCode" : 95111 }
333	

In the last row, ADDRESS is null because the data does not match the path expression. There are options to obtain other behaviors for empty and error conditions.

2.1.7 JSON_TABLE function

JSON_TABLE is a table function that is invoked in the FROM clause of a query to generate a relational table from a JSON value. As a simple example, to extract the scalar values of name and ZIP code from each JSON document, the following query can be used:

```
SELECT T.Id, Jt.Name, Jt.Zip
FROM T,
  JSON_TABLE ( T.Jcol, 'lax $'
    COLUMNS (
      Name VARCHAR ( 30 )
           PATH 'lax $.Name'
      Zip  VARCHAR ( 5 ) PATH
           'lax $.address.postalCode'
      )
   ) AS Jt
```

In this example, the first path expression 'lax $' is the "row pattern" used to locate rows. The path expression here is the simplest possible, just $, meaning that there is no navigation within a JSON document; however, if the row data were deeply nested within a JSON document, then the row pattern would be more complicated.

The example defines two output columns, NAME and ZIP. Each output column has its own PATH clause, specifying the "column pattern" that is used to navigate within a row to locate the data for a column.

The example has the following results on the sample data:

ID	NAME	ZIP
111	John Smith	10021
222	Peter Walker	95111
333	James Lee	

JSON_TABLE also allows unnesting of (even deeply) nested JSON objects/arrays in one invocation by using a nested COLUMNS clause, as the next example illustrates. This query will return the name and phone number and type for each person:

```
SELECT T.Id, Jt.Name, Jt.Type,
       Jt.Number
FROM T,
  JSON_TABLE ( T.Jcol, 'lax $'
    COLUMNS
     ( Name VARCHAR ( 30 )
            PATH 'lax $.Name',
       NESTED PATH
           'lax $.phoneNumber[*]'
       COLUMNS
         ( Type VARCHAR ( 10 )
                PATH 'lax $.type',
           Number VARCHAR ( 12 )
                PATH 'lax $.number' )
     )
  ) AS Jt
```

The preceding example has an outer row pattern 'lax $' and within that, a nested row pattern 'lax $.phoneNumber[*]'. The nested row pattern uses the wildcard array element accessor [*] to iterate over all elements of the phoneNumber array. Thus it is possible to flatten hierarchical data. The first column Name is found at the outer level of the hierarchy, whereas the nested columns Type and Number are found in the inner level of the hierarchy. The result on the sample data is:

ID	NAME	TYPE	NUMBER
111	John Smith	home	212 555-1234
111	John Smith	fax	646 555-4567
222	Peter Walker	home	408 555-9876
222	Peter Walker	office	650 555-2468
333	James Lee		

The last row has no phone number. If one wanted to include only those JSON documents that have a phoneNumber member, the following WHERE clause could be appended to the previous query:

```
WHERE JSON_EXISTS ( T.Jcol,
                    'lax $.phoneNumber' )
```

In the sample data, this WHERE clause would filter out the row whose ID is 333.

2.1.8 Structural-inspection methods

Because the structure of JSON data may not be known a priori and/or vary from one JSON value to the next, the SQL/JSON path language provides methods that allow for the inspection of the structural aspects of a JSON value. These methods are:

— keyvalue, which returns an SQL/JSON object containing three members for the key, the bound value, and an ID uniquely identifying the containing input object for each member of the input SQL/JSON object.

— `type`, which returns “object”, “array”, “string”, “number”, etc. corresponding to the actual type of the SQL/JSON item.

— `size`, which returns the number of elements, if the input SQL/JSON item is an array; otherwise it returns 1.

For example, to retain only arrays of size 2 or more, one might use:

```
strict $.* ? ( @.type() == "array"
            && @.size() > 1 )
```

2.2 SQL/JSON constructor functions

SQL/JSON constructor functions use values of SQL types and produce JSON (either JSON objects or JSON arrays) represented in SQL character or binary string types.

The functions are: `JSON_OBJECT`, `JSON_ARRAY`, `JSON_OBJECTAGG`, and `JSON_ARRAYAGG`. The first two are scalar functions, whereas the latter two are aggregate functions. As with other SQL functions/expressions, these can be arbitrarily nested. This supports the construction of JSON data of arbitrary complexity.

For example, given the well-known `Employees` and `Departments` tables, one can construct a JSON object for each department that contains all employees and their salary, sorted by increasing salary using this query:

```
SELECT JSON_OBJECT
 ( KEY 'department' VALUE D.Name,
   KEY 'employees'
    VALUE JSON_ARRAYAGG
          ( JSON_OBJECT
            ( KEY 'employee'
               VALUE E.Name,
              KEY 'salary'
               VALUE E.Salary )
             ORDER BY E.Salary ASC )
 ) AS Department
FROM Departments D, Employees E
WHERE D.Dept_id = E.Dept_id
GROUP BY D.Name
```

with results that might look like this:

DEPARTMENT
{ "department" : "Sales", "employees" : [{ "employee" : "James", "salary" : 7000 }, { "employee" : "Rachel", "salary" : 9000 }, { "employee" : "Logan", "salary" : 10000 }] }
…

3. POLYMORPHIC TABLE FUNCTIONS

The SQL standard prior to the 2016 release had only support for monomorphic table functions, *i.e.*, the definition of both the output table and the set of input parameters were fixed at function creation time. With the specification of polymorphic table functions (PTFs), SQL:2016 includes a very powerful enhancement to table functions. With this feature, the RDBMS is able to evaluate custom functionality closer to the data. For example, the MapReduce paradigm could be implemented using PTFs.

A polymorphic table function is a function that may have generic table input parameter(s) whose row type(s) may be unknown at creation time. The PTF may return a table whose row type also may be unknown when the function is created. The row type of the result may depend on the function arguments or on the row type(s) of the input table(s) in the invocation of the PTF. When a PTF is invoked in a query, the RDBMS and the PTF interact through a family of one to four SQL-invoked procedures. These procedures are called the PTF component procedures[7].

In the next sections, we describe these four component procedures and give two examples of user-defined PTFs. Due to space restrictions, we cannot cover all PTF features. For a detailed description of all aspects of PTFs (including the different perspectives of query author, PTF author, and RDBMS developer) the interested reader is referred to [16].

3.1 PTF Component Procedures

There are one to four PTF component procedures:

1. “describe”: The PTF describe component procedure is called once during compilation of the query that invokes the PTF. The primary task of the PTF describe component procedure is to determine the row type of the output table. This component procedure receives a description of the input tables and their ordering (if any) as well as any scalar input arguments that are compile-time constants. This component procedure is optional if all result columns are defined statically in the `CREATE FUNCTION` statement or if the PTF has only result columns that are passed through unchanged from the input table(s); otherwise it is mandatory.

2. “start”: The PTF start component procedure is called at the start of the execution of the PTF to allocate any resources that the RDBMS does not provide. This procedure is optional.

[7] SQL:2016 uses the four PTF component procedures as a specification vehicle. A conforming implementation may substitute this interface with an implementation-defined API.

3. “fulfill”: The PTF fulfill component procedure is called during the execution to deliver the output table by “piping” rows to the RDBMS. This is the component procedure that reads the contents of the input table(s) and generates the output table. This procedure is mandatory.

4. “finish”: The PTF finish component procedure is called at the end of the execution to deallocate any resources allocated by the PTF start component procedure. This procedure is optional.

3.2 Execution model

The SQL standard defines the run-time execution of a PTF using an abstraction called a virtual processor, defined as a processing unit capable of executing a sequential algorithm. Using techniques such as multiprocessing, a single physical processor might host several virtual processors. Virtual processors may execute independently and concurrently, either on a single physical processor or distributed across multiple physical processors. There is no communication between virtual processors. The RDBMS is responsible for collecting the output on each virtual processor; the union of the output from all virtual processors is the result of the PTF. The virtual processor abstraction is the standard’s way of permitting but not requiring parallelization of PTF execution.

3.3 Examples

This section illustrates the value of PTFs using a couple of examples. The first one is a simple example introducing the polymorphic nature of a PTF. The second one illustrates a variety of options including multiple input tables with different input semantics.

3.3.1 CSV reader table function

Consider a file with a comma-separated list of values (CSV file). The first line of the file contains a list of column names, and subsequent lines of the file contain the actual data. A PTF called `CSVreader` was created to read a CSV file and provide its data as a table in the `FROM` clause of a query. The effective signature of the PTF is:

```
FUNCTION CSVreader (
     File VARCHAR ( 1000 ),
     Floats DESCRIPTOR DEFAULT NULL,
     Dates DESCRIPTOR DEFAULT NULL )
RETURNS TABLE
NOT DETERMINISTIC
CONTAINS SQL
```

This signature has two parameter types that are distinctive to PTFs. (a) `DESCRIPTOR` is a type that is capable of describing a list of column names, and optionally for each column name, a data type. (b) `TABLE` denotes the generic table type, a type whose value is a table. The row type of the table is not specified, and may vary depending on the invocation of the PTF. Here, the `TABLE` specification specifies a generic table output of `CSVreader`. The row type is unknown at creation time and this is characteristic of a PTF.

A user reference guide accompanying a PTF will need to describe the semantics of the input parameters and what the output will be. For example, here `File` is the name of a file that contains the comma-separated values which are to be converted to a table. The first line of the file contains the names of the resulting columns. Succeeding lines contain the data. Each line after the first will result in one row of output, with column names as determined by the first line of the input. In the example above, `Floats` and `Dates` are PTF descriptor areas, which provide a list of the column names that are to be interpreted numerically and as dates, respectively; the data types of all other columns will be `VARCHAR`. With that information a query such as the following can be written:

```
SELECT *
FROM TABLE
    ( CSVreader (
         File => 'abc.csv',
         Floats => DESCRIPTOR
          ( "principal", "interest" )
         Dates => DESCRIPTOR
          ( "due_date" ) ) ) AS S
```

In the `FROM` clause, the `TABLE` operator introduces the invocation of a table function. A table function might be either a conventional (monomorphic) table function or a PTF. In this case, because `CSVreader` is declared with return type `TABLE`, this is a PTF invocation. This invocation says that `CSVreader` should open the file called abc.csv. The list of output column names is found in the first line of the file. Among these column names, there must be columns named `principal` and `interest`, which should be interpreted as numeric values, and a column named `due_date` which should be interpreted as a date. During the compilation of this query, the RDBMS will call the PTF describe component procedure and provide this information to the component procedure. In return, the component procedure will provide the RDBMS with the row type of the result table.

The component procedures are SQL procedures that are specified as a part of the definition of a PTF. For example:

```
CREATE FUNCTION CSVreader (
      File VARCHAR(1000),
      Floats DESCRIPTOR DEFAULT NULL,
      Dates DESCRIPTOR DEFAULT NULL )
RETURNS TABLE
```

```
NOT DETERMINISTIC CONTAINS SQL
PRIVATE DATA ( FileHandle INTEGER )
DESCRIBE WITH PROCEDURE
         CSVreader_describe
START WITH PROCEDURE
         CSVreader_start
FULFILL WITH PROCEDURE
         CSVreader_fulfill
FINISH WITH PROCEDURE
         CSVreader_finish
```

The procedures `CSVreader_describe`, `CSVreader_start`, `CSVreader_fulfill`, and `CSVreader_finish` are SQL stored procedures and can take advantage of the existing procedural language, dynamic SQL, and other existing SQL capabilities.

3.3.2 User Defined Join table function

The following example demonstrates a variety of options that can be specified in a PTF. It is a function that has input tables and also introduces options related to input table semantics (row or set semantics, keep or prune when empty) and the use of pass-through columns to flow data unaltered to the result table.

The PTF `UDJoin` performs a custom user-defined join. It takes two input tables, `T1` and `T2`, and matches rows according to some user defined join criterion that may not be built into the database system. The PTF has the following signature:

```
CREATE FUNCTION UDJoin
 ( T1 TABLE PASS THROUGH
            WITH SET SEMANTICS
            PRUNE WHEN EMPTY,
   T2 TABLE PASS THROUGH
            WITH SET SEMANTICS
            KEEP WHEN EMPTY
 ) RETURNS ONLY PASS THROUGH
```

The `RETURNS ONLY PASS THROUGH` syntax declares that the PTF does not generate any columns of its own; instead the only output columns are passed through from input columns.

`WITH SET SEMANTICS` is specified when the outcome of the function depends on how the data is partitioned. A table should be given set semantics if all rows of a partition should be processed on the same virtual processor. In this example, the entire table T2 is sent to the virtual processors.

`WITH ROW SEMANTICS` specified on an input table means that the result of the PTF is decided on a row-by-row basis for this input table. This is specified if the PTF does not care how rows are assigned to virtual processors. Only tables with set semantics may be partitioned and/or ordered.

The `KEEP WHEN EMPTY` option implies that the PTF could generate result rows even if the input table (in this example, `T2`), is empty. The result rows are based on rows from the other input table `T1`. `T1` is specified with `PRUNE WHEN EMPTY`, meaning that there is no output when the input table is empty. This example is analogous to a left outer join.

The `UDJoin` PTF can be invoked in a query like this:

```
SELECT E.*, D.*
FROM TABLE
     ( UDJoin (
           T1 => TABLE (Emp) AS E
                 PARTITION BY Deptno,
           T2 => TABLE (Dept) AS D
                 PARTITION BY Deptno
                 ORDER BY Tstamp ) )
```

In this example, both input tables have set semantics, which permits the use of `PARTITION BY` and `ORDER BY` clauses. `PARTITION BY` says that the input table is partitioned on a list of columns; each partition must be processed on a separate virtual processor. In this example, since there are two partitioned tables, the RDBMS must in fact create the cross product of the partitions of the two tables, with a virtual processor for each combination of partitions. (In the absence of `PARTITION BY`, a table with set semantics constitutes a single partition.) The second input table is also ordered; the RDBMS must sort the rows of each partition prior to passing them to the fulfill component procedure executing on any virtual processor.

Consider the following variation of the same query:

```
SELECT E.*, D.*
FROM TABLE
    ( UDJoin
     ( T1 => TABLE (Emp) AS E
             PARTITION BY Deptno,
       T2 => TABLE (Dept) AS D
             PARTITION BY Deptno
             ORDER BY Tstamp
             COPARTITION (Emp,Dept)))
```

Here, the `COPARTITION` clause allows each virtual processor to avoid the cross product as in the earlier example and collocates the corresponding values in the `Deptno` columns from the two tables in the same virtual processor.

4. ROW PATTERN RECOGNITION

Row Pattern Recognition (RPR) can be used to search an ordered partition of rows for matches to a regular expression. RPR can be supported in either the `FROM` clause or the `WINDOW` clause. This article will discuss

RPR in the FROM clause; RPR in the WINDOW clause uses much the same syntax and semantics.

RPR in the FROM clause uses the keyword MATCH_RECOGNIZE as a postfix operator on a table, called the *row pattern input table*. MATCH_RECOGNIZE operates on the row pattern input table and produces the *row pattern output table* describing the matches to the pattern that are discovered in the row pattern input table. There are two principal variants of MATCH_RECOGNIZE:

— ONE ROW PER MATCH, which returns a single summary row for each match of the pattern (the default).

— ALL ROWS PER MATCH, which returns one row for each row of each match.

The following example illustrates MATCH_RECOGNIZE with the ONE ROW PER MATCH option. Let Ticker (Symbol, Tradeday, Price) be a table with three columns representing historical stock prices. Symbol is a character column, Tradeday is a date column and Price is a numeric column. It is desired to partition the data by Symbol, sort it into increasing Tradeday order, and then detect maximal "V" patterns in Price: a strictly falling price, followed by a strictly increasing price. For each match to a V pattern, it is desired to report the starting price, the price at the bottom of the V, the ending price, and the average price across the entire pattern. The following query may be used to perform this pattern matching problem:

```
SELECT
   M.Symbol,  /* ticker symbol */
   M.Matchno, /* match number */
   M.Startp,  /* starting price */
   M.Bottomp, /* bottom price */
   M.Endp,    /* ending price */
   M.Avgp     /* average price */
FROM Ticker
  MATCH_RECOGNIZE (
    PARTITION BY Symbol
    ORDER BY Tradeday
    MEASURES
      MATCH_NUMBER() AS Matchno,
      A.Price        AS Startp,
      LAST (B.Price) AS Bottomp,
      LAST (C.Price) AS Endp,
      AVG (U.Price)  AS Avgp
    ONE ROW PER MATCH
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (A B+ C+)
    SUBSET U = (A, B, C)
    DEFINE
    /* A defaults to True,
          matches any row */
      B AS B.Price < PREV (B.Price),
      C AS C.Price > PREV (C.Price)
  ) AS M
```

In this example:

— Ticker is the row pattern input table.

— MATCH_RECOGNIZE introduces the syntax for row pattern recognition.

— PARTITION BY specifies how to partition the row pattern input table. If omitted, the entire row pattern input table constitutes a single partition.

— ORDER BY specifies how to order the rows within partitions.

— MEASURES specifies *measure columns*, whose values are calculated by evaluating expressions related to the match. The first measure column uses the nullary function MATCH_NUMBER(), whose value is the sequential number of a match within a partition. The third and fourth measure columns use the LAST operation, which obtains the value of an expression in the last row that is mapped by a row pattern match to a row pattern variable. LAST is one of many row pattern navigation operations, which may be used to navigate to specific rows of interest within a match.

— ONE ROW PER MATCH specifies that the result, the *row pattern output table*, will have a single row for each match that is found in the row pattern input table; each output row has one column for each partitioning column and one column for each measure column.

— AFTER MATCH SKIP specifies where to resume looking for the next row pattern match after successfully finding a match. In this example, PAST LAST ROW specifies that pattern matching will resume after the last row of a successful match.

— PATTERN specifies the row pattern that is sought in the row pattern input table. A row pattern is a regular expression using primary row pattern variables. In this example, the row pattern has three primary row pattern variables (A, B, and C). The pattern is specified as (A B+ C+) which will match a single A followed by one or more Bs followed by one or more Cs. An extensive set of regular expression specifications are supported.

— SUBSET defines the union row pattern variable U as the union of A, B, and C.

— DEFINE specifies the Boolean condition that defines a primary row pattern variable; a row must satisfy the Boolean condition in order to be mapped to a particular primary row pattern variable. This example uses PREV, a row pattern navigation operation that evaluates an expression in the previous row. In this

example, the row pattern variable A is undefined, in which case any row can be mapped to A.

— AS M defines the range variable M to associate with the row pattern output table.

Here is some sample data for Ticker, having one match to the pattern in the example (the mapping of rows to primary row pattern variables is shown in the last column):

Symbol	Tradeday	Price	Mapped to
XYZ	2009-06-08	50	
XYZ	2009-06-09	60	A
XYZ	2009-06-10	49	B
XYZ	2009-06-11	40	B
XYZ	2009-06-12	35	B
XYZ	2009-06-13	45	C
XYZ	2009-06-14	45	

Here is the row of the row pattern output table generated by the match shown above:

Symbol	Matchno	Startp	Bottomp	Endp	Avgp
XYZ	1	60	35	45	45.8

Due to space restrictions, we cannot cover all RPR features. For a detailed description of the RPR functionality the interested reader is referred to [14].

5. ADDITIONAL FUNCTIONALITY

5.1 Default values and named arguments for SQL-invoked functions

SQL:2011 allowed for a parameter of an SQL-invoked procedure to have a default value and thus be optional when invoking the procedure. A companion enhancement is invoking a procedure using named arguments [18]. SQL:2016 extends this functionality to cover SQL-invoked functions as well. For example, a function that computes the total compensation as the sum of the base salary and the bonus (where the bonus by default is 1000) can be defined like this:

```
CREATE FUNCTION Total_comp (
   Base_sal DECIMAL(7,2),
   Bonus DECIMAL(7,2) DEFAULT 1000.00
   ) RETURNS DECIMAL(8,2)
     LANGUAGE SQL
     RETURN Base_sal + Bonus;
```

This function can now be invoked in different ways:

— Passing all arguments by position:

```
Total_comp(9000.00, 1000.00)
```

— Passing all non-defaulted arguments by position:

```
Total_comp(9000.00)
```

— Passing all arguments by name (in this case the order of the arguments does not need to match the order of the parameters in the function signature):

```
Total_comp(Bonus=>1000.00,
           Base_sal=>9000.00)
```

— Passing all non-defaulted arguments by name:

```
Total_comp(Base_sal=>9000.00)
```

All of these invocations return the same result (10000.00). It should be clear that the greatest benefit of named and defaulted arguments can be realized when the parameter list is long and many parameters have useful defaults. Specifying some argument values by position and other arguments by name within the same invocation is not supported.

5.2 Additional built-in functions

SQL:2016 adds support for additional scalar mathematical built-in functions including trigonometric and logarithm functions. The trigonometric functions are sine, cosine, tangent, and their hyperbolic and inverse counterparts. Besides the existing natural logarithm function, SQL:2016 now supports a general logarithm function (where the user can specify an arbitrary value for the base) and a common logarithm function (with the base fixed at 10).

LISTAGG is a new aggregate function that allows concatenating character strings over a group of rows.

6. FUTURES

The SQL standards committee is currently working on additional expansions in three areas, support for multi-dimensional arrays, support for streaming data, and support for property graphs.

The work on multi-dimensional arrays (aka SQL/MDA) is well under way and will be completed by the end of 2018. This new incremental part adds a multi-dimensional array type so that instances of a multi-dimensional array can be stored in a column of a table and operations can be executed close to the data in the RDBMS.

The SQL committee has begun investigating requirements for streaming data and property graphs in the context of SQL.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge the following people, who at one point or another in one form or another have contributed to the standardization of the functionality presented in this paper: Jim Melton, Fatma Ozcan, Hamid Pirahesh, Doug McMahon, Shashaanka Agrawal, Ivan Bowman, Karl Schendel, Janhavi Rajagopal, Kathy McKnight, Hans Zeller, Dan

Pasco, Andy Witkowski, Chun-chieh Lin, Lei Sheng, and Taoufik Abdellatif.

The authors would like to thank the following people for reviewing earlier drafts of this paper and providing valuable feedback: Jim Melton, Hermann Baer, Mark Anderson, Andy Witkowski, and Claire McFeely.

8. REFERENCES

[1] ISO/IEC 9075-1:2016, *Information technology — Database languages — SQL* — Part 1: Framework (SQL/Framework)

[2] ISO/IEC 9075-2:2016, *Information technology — Database languages — SQL* — Part 2: Foundation (SQL/Foundation)

[3] ISO/IEC 9075-3:2016, *Information technology — Database languages — SQL* — Part 3: Call-Level Interface (SQL/CLI)

[4] ISO/IEC 9075-4:2016, *Information technology — Database languages — SQL* — Part 4: Persistent stored modules (SQL/PSM)

[5] ISO/IEC 9075-9:2016, *Information technology — Database languages — SQL* — Part 9: Management of External Data (SQL/MED)

[6] ISO/IEC 9075-10:2016, *Information technology — Database languages — SQL* — Part 10: Object language bindings (SQL/OLB)

[7] ISO/IEC 9075-11:2016, *Information technology — Database languages — SQL* — Part 11: Information and definition schemas (SQL/Schemata)

[8] ISO/IEC 9075-13:2016, *Information technology — Database languages — SQL* — Part 13: SQL Routines and types using the Java programming language (SQL/JRT)

[9] ISO/IEC 9075-14:2016, *Information technology — Database languages — SQL* — Part 14: XML-Related Specifications (SQL/XML)

[10] ISO/IEC TR 19075-1:2011, *Information technology — Database languages — SQL Technical Reports* — Part 1: XQuery Regular Expression Support in SQL, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[11] ISO/IEC TR 19075-2:2015, *Information technology — Database languages — SQL Technical Reports* — Part 2: SQL Support for Time-Related Information, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[12] ISO/IEC TR 19075-3:2015, *Information technology — Database languages — SQL Technical Reports* — Part 3: SQL Embedded in Programs using the Java™ programming language, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[13] ISO/IEC TR 19075-4:2015, *Information technology — Database languages — SQL Technical Reports* — Part 4: SQL with Routines and types using the Java™ programming language, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[14] ISO/IEC TR 19075-5:2016, *Information technology — Database languages — SQL Technical Reports* — Part 5: Row Pattern Recognition in SQL, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[15] ISO/IEC TR 19075-6:2017, *Information technology — Database languages — SQL Technical Reports* — Part 6: SQL support for JavaScript Object Notation (JSON), http://standards.iso.org/ittf/PubliclyAvailableStandards/

[16] ISO/IEC TR 19075-7:2017, *Information technology — Database languages — SQL Technical Reports* — Part 7: Polymorphic table functions in SQL, http://standards.iso.org/ittf/PubliclyAvailableStandards/

[17] Krishna Kulkarni and Jan-Eike Michels, "Temporal features in SQL:2011", SIGMOD Record Vol. 41 No. 3, September 2012, https://sigmodrecord.org/publications/sigmodRecord/1209/pdfs/07.industry.kulkarni.pdf

[18] Fred Zemke, "What's new in SQL:2011", SIGMOD Record, Vol. 41, No. 1, March 2012, https://sigmodrecord.org/publications/sigmodRecord/1203/pdfs/10.industry.zemke.pdf

[19] Andrew Eisenberg and Jim Melton, "Advancements in SQL/XML", SIGMOD Record Vol. 33 No. 3, September 2004, https://sigmodrecord.org/publications/sigmodRecord/0409/11.JimMelton.pdf

[20] Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels, and Fred Zemke, "SQL:2003 has been published", SIGMOD Record Vol. 33 No. 1, March 2004, https://sigmodrecord.org/publications/sigmodRecord/0403/E.JimAndrew-standard.pdf

[21] Jim Melton, Jan-Eike Michels, Vanja Josifovski, Krishna Kulkarni, Peter Schwarz, Kathy Zeidenstein, "SQL and Management of External Data", SIGMOD Record Vol. 30 No. 1, March 2001, https://sigmodrecord.org/publications/sigmodRecord/0103/JM-Sta.pdf

[22] Andrew Eisenberg and Jim Melton, "SQL:1999, formerly known as SQL3", SIGMOD Record Vol. 28 No. 1, March 1999, https://sigmodrecord.org/publications/sigmodRecord/9903/standards.pdf.gz

[23] Andrew Eisenberg, "New Standard for Stored Procedures in SQL", SIGMOD Record Vol 25 No. 4, Dec.1996, https://sigmodrecord.org/issues/96-12/sqlpsm.ps

[24] Internet Engineering Task Force, RFC 7159, The JavaScript Object Notation (JSON) Data Interchange Format, March 2014, https://tools.ietf.org/html/rfc7159

The Complex Event Recognition Group

Elias Alevizos Alexander Artikis Nikos Katzouris Evangelos Michelioudakis
Georgios Paliouras
Institute of Informatics & Telecommunications,
National Centre for Scientific Research (NCSR) Demokritos, Athens, Greece
{alevizos.elias, a.artikis, nkatz, vagmcs, paliourg}@iit.demokritos.gr

ABSTRACT

The *Complex Event Recognition* (CER) group is a research team, affiliated with the National Centre of Scientific Research "Demokritos" in Greece. The CER group works towards advanced and efficient methods for the recognition of complex events in a multitude of large, heterogeneous and interdependent data streams. Its research covers multiple aspects of complex event recognition, from efficient detection of patterns on event streams to handling uncertainty and noise in streams, and machine learning techniques for inferring interesting patterns. Lately, it has expanded to methods for forecasting the occurrence of events. It was founded in 2009 and currently hosts eight senior and junior researchers, working regularly with under-graduate students.

1. INTRODUCTION

The proliferation of devices that work in real-time, constantly producing data streams, has led to a paradigm shift with respect to what is expected from a system working with massive amounts of data. The dominant model for processing large-scale data was one that assumed a relatively fixed database/knowledge base, i.e., it assumed that the operations of updating existing records/facts and inserting new ones were infrequent. The user of such a system would then pose queries to the database, without very strict requirements in terms of latency.

While this model is far from being rendered obsolete (on the contrary), a system aiming to extract actionable knowledge from continuously evolving streams of data has to address a new set of challenges and satisfy a new set of requirements. The basic idea behind such a system is that it is not always possible, or even desirable, to store every bit of the incoming data, so that it can be later processed. Rather, the goal is to make sense out of these streams of data, without having to store them. This is done by defining a set of queries/patterns, continuously applied to the data streams. Each such pattern includes a set of temporal constraints and, possibly, a set of spatial constraints, expressing a composite or complex event of special significance for a given application. The system must then be efficient enough so that instances of pattern satisfaction can be reported to a user with minimal latency. Such systems are called Complex Event Recognition (CER) systems [6, 7, 2].

CER systems are widely adopted in contemporary applications. Such applications are the recognition of attacks in computer network nodes, human activities on video content, emerging stories and trends on the Social Web, traffic and transport incidents in smart cities, fraud in electronic marketplaces, cardiac arrhythmias and epidemic spread. Moreover, Big Data frameworks, such as Apache Storm, Spark Streaming and Flink, have been extending their stream processing functionality by including implementations for CER.

There are multiple issues that arise for a CER system. As already mentioned, one issue is the requirement for minimal latency. Therefore, a CER system has to employ highly efficient reasoning mechanisms, scalable to high-velocity streams. Moreover, pre-processing steps, like data cleaning, have to be equally efficient, otherwise they constitute a luxury that a CER system cannot afford. In this case, the system must be able to handle noise. This may be a requirement, even if perfectly clean input data is assumed, since domain knowledge is often insufficient or incomplete. Hence, the patterns defined by the users may themselves carry a certain degree of uncertainty. Moreover, it is quite often the case that such patterns cannot be provided at all, even by domain experts. This poses a further challenge of how to apply machine learning techniques in order to extract patterns from streams before a CER system can actually run with them. Standard machine learning techniques are not always directly applicable, due to the size and variability of the training set. As a result, machine learning techniques must work in an online fashion. Finally, one often needs

to move beyond detecting instances of pattern satisfaction into *forecasting* when a pattern is likely to be satisfied in the future.

Our CER group[1] at the National Centre for Scientific Research (NCSR) "Demokritos", in Athens, Greece, has been conducting research on complex event recognition for the past decade, and has developed a number of novel algorithms and open-source software tools. NCSR "Demokritos" is the largest multi-disciplinary research center in Greece, with expertise and infrastructure in the fields of Informatics and Telecommunications, Nanotechnology, Energy & Environment, Biosciences, and Particle and Nuclear Science. The Institute of Informatics and Telecommunications, in particular, focuses on research and development in the areas of Intelligent Systems, Telecommunications, Networks and Web Technologies. The CER group is one of the six groups the Software & Knowledge Engineering Lab of the Institute of Informatics & Telecommunications. In what follows, we sketch the approaches that we have proposed as part of the CER group and present some indicative results.

2. COMPLEX EVENT RECOGNITION

Numerous CER systems have been proposed in the literature [6, 7]. Recognition systems with a logic-based representation of complex event (CE) patterns, in particular, have been attracting attention since they exhibit a formal, declarative semantics [2]. We have been developing an efficient dialect of the Event Calculus, called 'Event Calculus for Run-Time reasoning' (RTEC) [4]. The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects [14]. CE patterns in RTEC identify the conditions in which a CE is initiated and terminated. Then, according to the law of inertia, a CE holds at a time-point T if it has been initiated at some time-point earlier than T, and has not been terminated in the meantime.

RTEC has been optimised for CER, in order to be scalable to high-velocity data streams. A form of caching stores the results of subcomputations in the computer memory to avoid unnecessary recomputations. A set of interval manipulation constructs simplify CE patterns and improve reasoning efficiency. A simple indexing mechanism makes RTEC robust to events that are irrelevant to the patterns we want to match and so RTEC can operate without data filtering modules. Finally, a 'windowing' mechanism supports real-time CER. One main motivation for RTEC is that it should remain efficient and scalable in applications where events arrive with a (variable) delay from, or are revised by, the underlying sensors: RTEC can update the intervals of the already recognised CEs, and recognise new CEs, when data arrive with a delay or following revision.

RTEC has been analysed theoretically, through a complexity analysis, and assessed experimentally in several application domains, including city transport and traffic management [5], activity recognition on video feeds [4], and maritime monitoring [18]. In all of these applications, RTEC has proven capable of performing real-time CER, scaling to large data streams and highly complex event patterns.

3. UNCERTAINTY HANDLING

CER applications exhibit various types of uncertainty, ranging from incomplete and erroneous data streams to imperfect CE patterns [2]. We have been developing techniques for handling uncertainty in CER by extending the Event Calculus with probabilistic reasoning. Prob-EC [20] is a logic programming implementation of the Event Calculus using the ProbLog engine [13], that incorporates probabilistic semantics into logic programming. Prob-EC is the first Event Calculus dialect able to deal with uncertainty in the input data streams. For example, Prob-EC is more resilient to spurious data than the standard (crisp) Event Calculus.

MLN-EC [21] is an Event Calculus implementation based on Markov Logic Networks (MLN)s [19], a framework that combines first-order logic with graphical models, in order to enable probabilistic inference and learning. CE patterns may be associated with weight values, indicating our confidence in them. Inference can then be performed regarding the time intervals during which CEs of interest hold. Like Prob-EC, MLN-EC increases the probability of a CE every time its initiating conditions are satisfied, and decreases this probability whenever its terminating conditions are satisfied, as shown in Figure 1. Moreover, in MLN-EC the domain-independent Event Calculus rules, expressing the law of inertia, may be associated with weight values, introducing probabilistic inertia. This way, the model is highly customisable, by tuning appropriately the weight values with the use of machine learning techniques, and thus achieves high predictive accuracy in a wide range applications.

The use of background knowledge about the task and the domain, in terms of logic (the Event Calculus), can make MLN-EC more robust to variations in the data. Such variations are very common in practice, particularly in dynamic environments, such as the ones encountered in CER. The

[1] `http://cer.iit.demokritos.gr/`

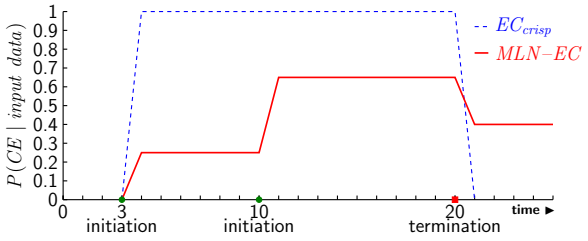


Figure 1: CE probability estimation in the Event Calculus. The solid line concerns a probabilistic Event Calculus, such as MLN-EC, while the dashed line corresponds to a crisp (non-probabilistic) version of the Event Calculus. Due to the law of inertia, the CE probability remains constant in the absence of input data. Each time the initiation conditions are satisfied (e.g., in time-points 3 and 10), the CE probability increases. Conversely, when the termination conditions are satisfied (e.g., in time-point 20), the CE probability decreases.

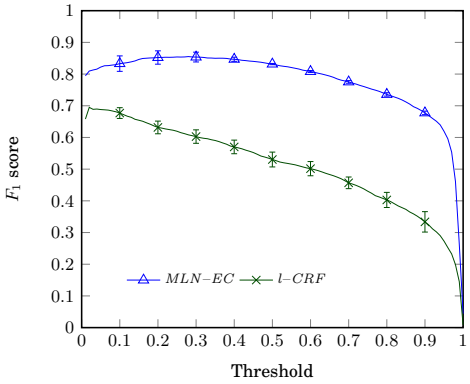


Figure 2: CER under uncertainty. F_1-score of MLN-EC and linear-chain CRFs for different CE acceptance thresholds.

common assumption made in machine learning that the training and test data share the same statistical properties is often violated in these situations. Figure 2, for example, compares the performance of MLN-EC against linear-chain Conditional Random Fields on a benchmark activity recognition dataset, where evidence is incomplete in the test set as compared to the training set.

4. EVENT PATTERN LEARNING

The manual authoring of CE patterns is a tedious and error-prone process. Consequently, the automated construction of such patterns from data is highly desirable. We have been developing supervised, online learning learning tools for constructing logical representations of CE patterns, from a single-pass over a relational data stream. OSLα [16] is such a learner for Markov Logic Networks (MLNs), formulating CE patterns in the form of MLN-EC theories. OSLα extends OSL [9] by exploiting a background knowledge in order to significantly constrain the search for patterns.

In each step t of the online procedure, a set of training examples $\mathcal{D}_t$ arrives containing input data along with CE annotation. $\mathcal{D}_t$ is used together with the already learnt hypothesis, if any, to predict the truth values of the CEs of interest. This is achieved by MAP (maximum a posteriori) inference. Given $\mathcal{D}_t$, OSLα constructs a hypergraph that represents the space of possible structures as graph paths. Then, for all incorrectly predicted CEs, the hypergraph is searched using relational pathfinding, for clauses supporting the recognition of these CEs. The paths discovered during the search are generalised into first-order clauses. Subsequently, the weights of the clauses that pass the evaluation stage are optimised using off-the-shelf online weight learners. Then, the weighted clauses are appended to the hypothesis and the procedure is repeated for the next set of training examples $\mathcal{D}_{t+1}$.

OLED [11] is an Inductive Logic Programming system that learns CE patterns, in the form of Event Calculus theories, in a supervised fashion and in a single pass over a data stream. OLED constructs patterns by first encoding a positive example from the input stream into a so-called *bottom rule*, i.e., a most-specific rule $\alpha \leftarrow \delta_1 \wedge \ldots \wedge \delta_n$, where α is an initiation or termination atom, and $\delta_1, \ldots, \delta_n$ are relational features expressing anything "interesting" as defined by the *language bias*. To learn a useful rule, OLED then searches within the space of rules that θ-*subsume* the bottom rule, i.e., rules that involve some of the δ_i's only. To that end, OLED starts from the most-general rule and gradually *specialises* it by adding δ_i's to its body, using a rule evaluation function to assess the quality of each generated specialisation. OLED's single-pass strategy is based on the *Hoeffding bound* [8], a statistical tool that allows to approximate the quality of a rule on the entire input using only a subset of the data.

We have evaluated OLED and OSLα on real datasets concerning activity recognition, maritime monitoring, credit card fraud detection, and traffic management in smart cities [11, 16, 3, 15, 12]. We have also compared OLED and OSLα to OSL [9], 'batch' structure learners requiring many passes over the data, and to hand-curated Event Calculus patterns (with optimised weight values). The results suggest that both OLED and OSLα can match the predictive accuracy of batch learners as well

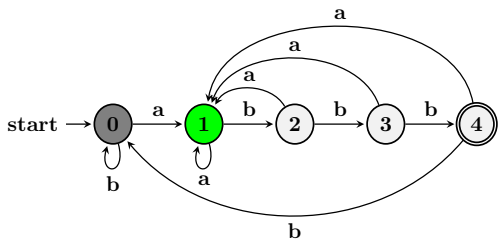
as that of hand-crafted patterns. Moreover, OLED and OSLα have proven significantly faster than both batch and online learners, making them more appropriate for large data streams.

5. EVENT FORECASTING

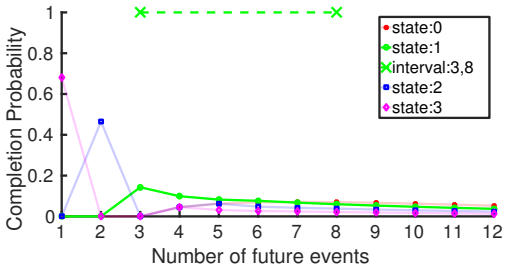
Forecasting over time-evolving data streams is a task that can be defined in multiple ways. There is a conceptual difference between forecasting and prediction, as the latter term is understood in machine learning, where the main goal is to "predict" the output of a function on previously unseen input data, even if there is no temporal dimension. In forecasting, *time* is a crucial component and the goal is to predict the temporally future output of some function or the occurrence of an event. Time-series forecasting is an example of the former case and is a field with a significant history of contributions. However, its methods cannot be directly transferred to CER, since it handles streams of (mostly) real-valued variables and focuses on forecasting relatively simple patterns. On the contrary, in CER we are also interested in categorical values, related through complex patterns and involving multiple variables. Our group has developed a method, where automata and Markov chains are employed in order to provide (future) time intervals during which a match is expected with a probability above a confidence threshold [1].

We start with a given pattern of interest, defining relations between events, in the form of a regular expression—i.e., using operators for *sequence*, *disjunction* and *iteration*. Our goal, besides detecting occurrences of this pattern, is also to estimate, at each new event arrival, the number of future events that we will need to wait for until the expression is satisfied, and thus a match be detected. A pattern in the form of a regular expression is first converted to a deterministic finite automaton (DFA) through standard conversion algorithms. We then construct a Markov chain that will be able to provide a probabilistic description of the DFA's run-time behavior, by employing Pattern Markov Chains (PMC) [17]. The resulting PMC depends both on the initial pattern and on the assumptions made about the statistical properties of the input stream—the order m of the assumed Markov process.

After constructing a PMC, we can use it to calculate the so-called *waiting-time* distributions, which can give us the probability of reaching a final state of the DFA in k transitions from now. To estimate the final forecasts, another step is required, since our aim is not to provide a single future point with the highest probability, but an interval in the form



(a) Deterministic Finite Automaton, state 1.



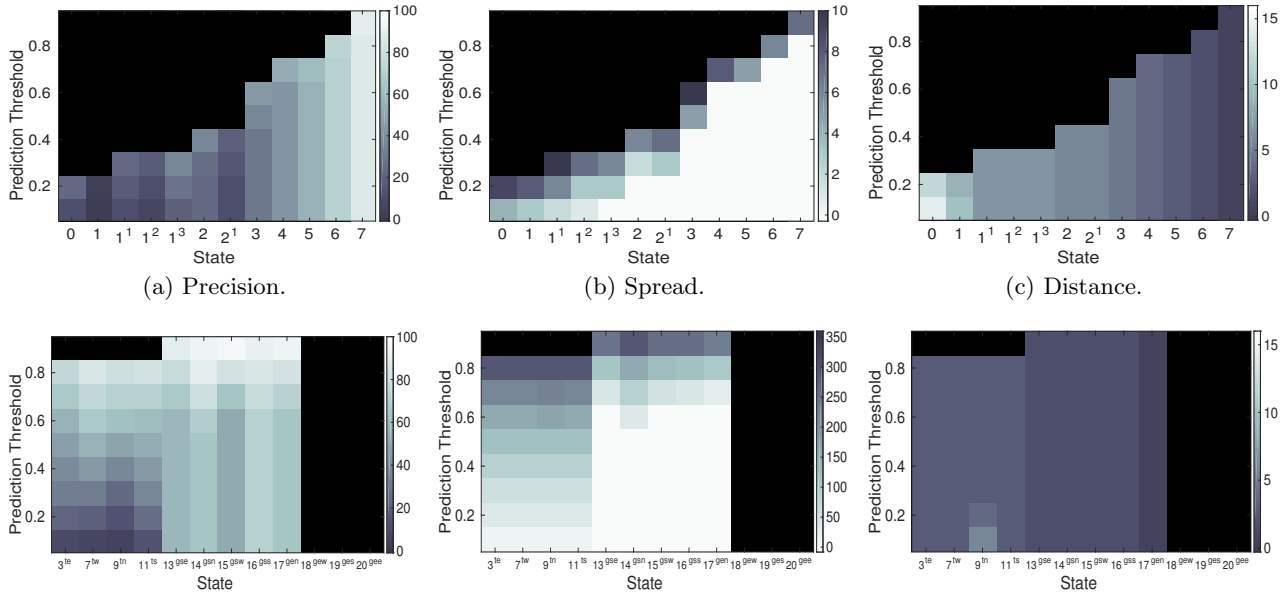
(b) Waiting-time distribution, state 1.

Figure 3: Event Forecasting. The event pattern requires that one event of type a is followed by three events of type b. $\theta_{fc} = 0.5$. For illustration, the x axis stops at 12 future events.

of I=($start$, end). The meaning of such an interval is that the DFA is expected to reach a final state sometime in the future between *start* and *end* with probability at least some constant threshold θ_{fc} (provided by the user). An example is shown in Figure 3, where the DFA in Figure 3a is in state 1, the *waiting-time* distributions for all of its non-final states are shown in Figure 3b, and the distribution, along with the forecast interval, for state 1 are shown in green.

Figure 4 shows results of our implementation on two real-world datasets from the financial and the maritime domains. In the former case, the goal was to forecast a specific case of credit card fraud, whereas in the latter it was to forecast a specific vessel manoeuver. Figures 4a and 4d show *precision* results (the percentage of forecasts that were accurate), where the y axes correspond to different values of the threshold θ_{fc}, and the x axes correspond to states of the PMC (more "advanced" states are to the right of the axis), i.e., we measure precision for the forecasts produced by each individual state. Similarly, Figures 4b and 4e are per-state plots for *spread* (the length of the forecast interval), and Figures 4c and 4f are per-state plots for *distance* (the temporal distance between the time a forecast is produced and the start of the forecast interval).

As expected, more "advanced" states produce forecasts with higher precision, smaller spread and dis-



(a) Precision. (b) Spread. (c) Distance.

(d) Precision. (e) Spread. (f) Distance.

Figure 4: Event forecasting for credit card fraud management (top) and maritime monitoring (bottom). The y axes correspond to different values of the threshold θ_{fc}. The x axes correspond to states of the PMC.

tance. However, there are cases where we can get earlier both high precision and low spread scores (see Figures 4d and 4e). This may happen when there exist strong probabilistic dependencies in the stream, e.g., when one event type is very likely (or very unlikely) to appear, given that the last event(s) is of a different event type. Our system can take advantage of such cases in order to produce high-quality forecasts early.

6. PARTICIPATION IN RESEARCH & INNOVATION PROJECTS

The CER group has been participating in several research and innovation projects.

SPEEDD[2] (Scalable Proactive Event-Driven Decision Making) was an FP7 EU-funded project, coordinated by the CER group, that developed tools for proactive analytics in Big Data applications. In SPEEDD, the CER group worked on credit card fraud detection and traffic management [3, 15], developing formal tools for highly scalable CER [4], and pattern learning [10, 16].

REVEAL[3] (REVEALing hidden concepts in social media) was an FP7 EU project that developed techniques for real-time knowledge extraction from social media. In REVEAL, the CER group developed a technique for *online* (single-pass) learning of event patterns under uncertainty [11].

AMINESS[4] (Analysis of Marine Information for Environmentally Safe Shipping) was a national project that developed a framework for maritime environmental safety and cost reduction.

Similarly, **datACRON**[5] (Big Data Analytics for Time Critical Mobility Forecasting) is an H2020 EU project that introduces novel methods for detecting threats and abnormal activity in very large fleets of moving entities, such as vessels and aircrafts. In datACRON, the CER group has been developing algorithms for highly efficient spatio-temporal pattern matching [18], complex event forecasting [1] and parallel, online event pattern learning [12], as well as user-friendly languages for manual pattern construction [22].

Track & Know (Big Data for Mobility & Tracking Knowledge Extraction in Urban Areas) is an H2020 EU-funded project that develops a new software framework increasing the efficiency of Big Data applications in the transport, mobility, motor insurance and health sectors. The CER team is responsible for the complex event recognition and forecasting technology of Track & Know.

7. COMMUNITY CONTRIBUTIONS

The CER group supports the research commu-

[2] http://speedd-project.eu/
[3] http://revealproject.eu/
[4] http://aminess.eu/
[5] http://www.datacron-project.eu/

nity at different levels; notably, by making available the proposed research methods as open-source solutions. The RTEC CER engine (see Section 2) is available as a monolithic Prolog implementation[6] and as a parallel Scala implementation[7]. The OLED system for online learning of event patterns (see Section 4) is also available as an open-source solution[8], both for single-core and parallel learning. OLED is implemented in Scala; both OLED and RTEC use the Akka actors library for parallel processing.

The OSLα online learner (see Section 4), along with MAP inference based on integer linear programming, and various weight optimisation algorithms (Max-Margin, CDA and AdaGrad), are contributed to LoMRF[9], an open-source implementation of Markov Logic Networks. LoMRF provides predicate completion, clausal form transformation and a parallel grounding algorithm which efficiently constructs the minimal Markov Random Field.

8. REFERENCES

[1] E. Alevizos, A. Artikis, and G. Paliouras. Event forecasting with pattern markov chains. In *DEBS*, pages 146–157. ACM, 2017.

[2] E. Alevizos, A. Skarlatidis, A. Artikis, and G. Paliouras. Probabilistic complex event recognition: A survey. *ACM Comput. Surv.*, 50(5):71:1–71:31, 2017.

[3] A. Artikis, N. Katzouris, I. Correia, C. Baber, N. Morar, I. Skarbovsky, F. Fournier, and G. Paliouras. A prototype for credit card fraud management: Industry paper. In *DEBS*, pages 249–260, 2017.

[4] A. Artikis, M. J. Sergot, and G. Paliouras. An event calculus for event recognition. *IEEE TKDE*, 27(4):895–908, 2015.

[5] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, and D. Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *EDBT*, pages 712–723, 2014.

[6] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

[7] N. Giatrakos, A. Artikis, A. Deligiannakis, and M. N. Garofalakis. Complex event recognition in the big data era. *PVLDB*, 10(12):1996–1999, 2017.

[8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *JASA*, 58(301):13–30, 1963.

[9] T. N. Huynh and R. J. Mooney. Online Structure Learning for Markov Logic Networks. In *ECML*, pages 81–96, 2011.

[10] N. Katzouris, A. Artikis, and G. Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.

[11] N. Katzouris, A. Artikis, and G. Paliouras. Online learning of event definitions. *TPLP*, 16(5-6):817–833, 2016.

[12] N. Katzouris, A. Artikis, and G. Paliouras. Parallel online learning of complex event definitions. In *ILP*. Springer, 2017.

[13] A. Kimmig, B. Demoen, L. D. Raedt, V. Costa, and R. Rocha. On the implementation of the probabilistic logic programming language ProbLog. *TPLP*, 11(2-3):235–262, 2011.

[14] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

[15] E. Michelioudakis, A. Artikis, and G. Paliouras. Online structure learning for traffic management. In *ILP*. Springer, 2016.

[16] E. Michelioudakis, A. Skarlatidis, G. Paliouras, and A. Artikis. Online structure learning using background knowledge axiomatization. In *ECML*, 2016.

[17] G. Nuel. Pattern Markov Chains: Optimal Markov Chain Embedding through Deterministic Finite Automata. *Journal of Applied Probability*, 2008.

[18] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2), 2017.

[19] M. Richardson and P. M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[20] A. Skarlatidis, A. Artikis, J. Filipou, and G. Paliouras. A probabilistic logic programming event calculus. *TPLP*, 15(2):213–245, 2015.

[21] A. Skarlatidis, G. Paliouras, A. Artikis, and G. A. Vouros. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic*, 16(2):11:1–11:37, 2015.

[22] C. Vlassopoulos and A. Artikis. Towards A simple event calculus for run-time reasoning. In *COMMONSENSE*, 2017.

[6] `https://github.com/aartikis/RTEC`
[7] `https://github.com/kontopoulos/ScaRTEC`
[8] `https://github.com/nkatzz/OLED`
[9] `https://github.com/anskarl/LoMRF`

DANTE: Data and Information Management Research at Nanyang Technological University

Sourav S Bhowmick, Xiaokui Xiao*

School of Computer Science and Engineering
Nanyang Technological University, Singapore

1. INTRODUCTION

The data management group at the Nanyang Technological University (DANTE) was founded in 2009 by the first author when the School of Computer Science and Engineering hired two young faculty members in this area. The group currently consists of three faculty members and more than twenty graduate students, research assistants, and postdocs. Our alumni include faculty members at the Chinese University of Hong Kong, National University of Singapore, University of New South Wales, and several researchers and engineers at Facebook, Google, eBay, Huawei, and other technology companies. The group's major funding is from the Ministry of Education in Singapore, National Research Foundation, and companies such as Huawei.

In DANTE, we subscribe to the policy of conducting research mostly in small groups. Typically, one or two faculty members work together with their students, staffs, and collaborators (if any). Our members bring in diverse strengths, some have penchant for inventing efficient and scalable solutions to existing data management problems whereas others are more inclined toward inventing novel problems and efficient solutions to address them. Our research is often multi-disciplinary in flavour, bridging data management and analytics with social science and biology. In particular, our research have been nominated as one of the best papers in venues such as SIGMOD 2015, ICDE 2015, and ICDE 2010. We have also received best paper award in ACM BCB 2011. The common thread running through our research is a focus on going beyond papers to build usable novel prototypes. Specifically, we have successfully demonstrated more than 20 novel research prototypes in top-tier data and information management conferences, which is the highest among all data management research groups in Australasia. In this article, we present a brief overview of the key research themes in our group; more details are available on our website at `http://www3.ntu.edu.sg/scse/dmal/dante/`.

*The authors are the founding members of DANTE. The second author is a faculty member of National University of Singapore since January 2018.

2. GRAPH DATA MANAGEMENT

Graphs are used to model data in a variety of domains such as biology, chemistry, and social science. Even we can model the relational data as a graph. This prevalence of graph-structured data has led us to pursue several research directions in this arena as follows.

2.1 Graph Query Processing

Querying graphs has emerged as an important research problem since the last decade. Our group has invented a suite of efficient and scalable techniques to support a variety of graph queries such as distance queries [15], subgraph enumeration [31], supergraph search [16], personalized PageRank queries [24, 51, 52], SimRank queries [38, 48], and reachability queries [14, 70]. In particular, our solutions for personalized PageRank and SimRank queries provide superior practical efficiency while providing strong theoretical guarantees in terms of accuracy and time complexity. In a different project, we questioned the longstanding assumption that a subgraph search query must be a *connected* graph. Such assumption typically demands users to have precise knowledge of the topological structure of what they are seeking. Unfortunately, due to the topological complexity of the underlying graph data, it is often unrealistic to assume that an end user is aware of the precise relationships between nodes in a query graph. This led us to invent a novel subgraph query processing framework called PANDA [59] that can efficiently support formulation and processing of *partial topology queries*. Such queries are disconnected query graphs comprising of two or more disjoint *connected query components*. This framework can also be used to address the problem of keyword search for graphs as a keyword can be considered as a single-node query component.

2.2 Human Interaction with Graphs

In our HINT project[1], we explore pioneering techniques and paradigms for visually interacting with graphs using queries. It is well-known that visual query interfaces (*i.e.,* GUI) enable interactive construction of graph queries without demanding knowledge of a graph query language from end users. In a classical visual graph querying framework, the visual query interface is "loosely-coupled" with the underlying query engine. Typically, a visual query interface is designed and implemented by leveraging principles from the human-computer interaction (HCI) area to enhance its usability. On the other hand, the query engine is realized using data management principles to ensure efficient and scalable execution of graph queries. Seldom there is any meaningful interactions between these two components *concurrently*. Consequently, when an end user is visually formulating a graph query, the underlying query engine remains idle as human interactions at the GUI level are rarely communicated to the query engine. The query engine is only invoked when the complete query has been formulated and the `Run` icon is clicked to execute it. We refer to this loose coupling of these two key components as *shallow integration*.

The visual graph query formulation process demonstrates two key characteristics. First, a query is gradually exposed to the underlying query engine during its construction. Second, it gives rise to GUI *latency* (*i.e.,* the time spent by a human to complete certain query formulation task such as construction of an edge). In this research, we crystallize "tight coupling" between visual graph query interface and query engine components by exploiting these features. Instead of the query engine being oblivious to human interactions in the GUI during visual query formulation, we "track" these interactions and process them judiciously during query formulation by exploiting the GUI latency. We refer to this tight coupling of the visual query interface and the query engine as *deep integration*.

In our group, we have explored a suite of novel techniques to realize deep integration. Specifically, in [63] we report a technique that leverages on partially constructed query information during query formulation to present opportune suggestions to end users toward completion of the query. These efforts realize deep integration between the visual query interface and underlying query engine by generating data-driven suggestions while a graph query is being visually formulated. In [26, 28, 29, 44], we realize deep integration by blending visual graph query formulation with its processing to prune false results and prefetch partial query results by exploiting the GUI latency, leading to superior system response time. We investigate a variety of graph queries (subgraph matching, subgraph similarity, and homomorphic queries) in this paradigm. In particular, these frameworks allow a user to execute a query fragment any time during query formulation and not wait until the entire query is visually formulated. Consequently, this paradigm is exploited by PICASSO [25] to realize exploratory search on graphs. Lastly, query performance study in a deep integration-based graph querying framework demands exhaustive user study due to tight coupling between human interactions and the underlying query engine. However, such user study is expensive and time-consuming. In [3], we present a framework called VISUAL that draws upon the literature in HCI and graph data management to simulate visual subgraph query construction process. This paves the way for automated performance study without requiring users.

2.3 Graph Analytics

Lastly, we have contributed efficient and scalable algorithms for addressing a variety of graph analytics problems. For instance, we have invented scalable algorithms for finding maximal cliques [11, 12], core and truss decomposition [13, 49], triangle listing [17], computation of maximum independent set [39], attributed graph clustering [60], and discovering frequent subgraph patterns using the MapReduce framework [36].

In particular, some of our research in graph analytics is multi-disciplinary in nature. For instance, we have explored the role of network analytics in biology. It is increasingly attractive to model biological systems from a broader, "systems" perspective instead of modeling its components in an isolated, reductionist manner [27]. The most well-known method to model biological systems in this manner is through *biological networks*. However, due to the complexity of such networks, it is challenging to uncover key system-wide properties and behaviors of a biological system from it. To this end, we have developed scalable techniques for discovering network motifs (*i.e.,* interaction patterns that recur throughout biological networks, much more often than in random networks) by leveraging modern hardware such as GPUs [37]. This work was nominated as one of the best in ICDE 2015. We have also developed novel techniques for *summarizing* static and dynamic biological networks [1] as well as predicting potential drug targets by ana-

[1] `http://www.ntu.edu.sg/home/assourav/research/hint/index.html`

lyzing dynamics of signaling networks [18]. In particular, our work on functional summarization of protein-protein interaction networks received the Best Paper Award in ACM BCB 2011 (flagship conference of the SIGBio group) [43].

3. SOCIAL DATA MANAGEMENT

In the social data management arena, DANTE members have primarily focused on two topics: social influence analysis in online social networks and social image exploration.

3.1 Social Influence Analysis

Our group has made significant contributions in social influence analysis especially in the context of the influence maximization problem. Given a social network G and a constant k, the influence maximization problem asks for k nodes in G that (directly and indirectly) influence the largest number of nodes under a pre-defined diffusion model. This problem originates from viral marketing, and it has been extensively studied in the literature since 2003. However, before 2014, there was a long-standing tension between the efficiency and accuracy of influence maximization algorithms. In particular, there exist a few methods that provide non-trivial approximation guarantees, but they require days to process even a small graph; meanwhile, methods with reasonable practical efficiency all rely on heuristics, due to which they fail to offer any worst-case accuracy assurance. We are the first to ease the tension by proposing Two-phase Influence Maximization (TIM) [46], an algorithm that runs in $O((k + l)(n + m) \log n/\epsilon^2)$ expected time, and returns a $(1-1/e-\epsilon)$-approximate solution to influence maximization, with at least $1/n^l$ probability. The time complexity of TIM is near-optimal, and it is empirically shown to be up to three orders of magnitude lower than any existing solution with non-trivial approximation guarantees. Subsequently, we develop an improvement of TIM [47] that retains its theoretical guarantees while improving its practical efficiency by up to another order of magnitude.

We also extended our research on influence maximization to competitive networks where several groups may simultaneously attempt to select seeds in the same network [33]. We proposed a framework called GETREAL that finds the best solution for each group, who are maximizing their influences, based on game theory. This work was one of the nominees for the best paper award in SIGMOD 2015.

In a separate project, we took the first systematic step to discover *k influential event organizers* from online social networks (*e.g., Meetup* (`www.meetup.com`) who are essential to the overall success of social events [20]. These event organizers comprise a small group of people who not only have the relevant skills or expertise that are required for an event but they are also able to influence largest number of people to actively contribute to it.

The aforementioned efforts as well as numerous other social influence research in the data management and data mining communities have largely stripped off social psychology of users in their solution design. For example, these efforts ignore *conformity* of people, which refers to a person's inclination to be influenced by others. Consequently, despite the great progress made in terms of algorithmic efficiency and scalability, existing techniques may not necessarily produce high quality results in practice. In our PAELLA project[2], we investigate the interplay between psychology and social influence in online social networks and devise novel social influence solutions that are psychology-aware. Specifically, we are the first to explore techniques that incorporate conformity in computing social influence and influence maximization solutions [34, 35].

3.2 Social Image Search Results Exploration

Due to increasing popularity of social image sharing platforms (*e.g., Flickr*), techniques to support *Tag-based Social Image Retrieval* (TAGIR) [32] for finding relevant high-quality images using keyword queries have generated tremendous research and commercial interests. Many TAGIR studies attempt to improve its search accuracy or diversify its search results so as to maximize the probability of satisfying users' search intentions. In our SIERRA project, we go beyond retrieval and ranking of social images by facilitating deeper understanding through explanation and exploration of the result images.

Why-not questions on search results. Traditional TAGIR systems fail to provide a systematic framework for end users to ask why certain images are not in the result set of a given query and provide an explanation for such missing results. However, as humans, such *why-not* questions [7] are natural when expected images are missing in the query results returned by a TAGIR system. This may be due to the following reasons. First, the desired images may be ranked very low in the search results because the same keyword query (*e.g.,* "rock") may express very different search intentions for different users. Second, the set of tags associated with images may be noisy and incomplete. Consequently, not all keywords mentioned in the search query may appear as tags in relevant images. Third, the query formulated by the user maybe too restrictive due

[2] `http://www.ntu.edu.sg/home/assourav/research/paella/index.html`

to the user's limited understanding of the data collection. Indeed, it will be helpful to users if they could simply pose a follow-up *why-not* question to the retrieval engine to seek an explanation for desired missing images and suggestions on how to retrieve them. Our group developed a novel system called WINE [2] to address this challenge. Specifically, it leverages on three explanation models that exploit *Wikipedia* to automatically generates explanation to a *why-not* question posed by a user and recommends refined query, if necessary, whose result may not only includes images related to the search query but also to the why-not question.

Search results summarization. Social image search engines often diversify the search results to match all possible aspects of a query in order to minimize the risk of completely missing out a user's search intent. An immediate aftermath of such results diversification strategy is that often the search results are not semantically or visually coherent. For example, the results of a search query "fly" may contain a medley of visually and semantically distinct objects and scenes (*i.e.,* concepts) such as parachutes, aeroplanes, insects, birds, and even the act of jumping. Consequently, a thumbnail view of query results fails to provide a bird eye view of different concepts present in it. Our PRISM [42] system addresses this challenge by constructing high quality summary of top-k social image search results based on *concept-preserving* and *visually coherent clusters* which maximally cover the result set. Each cluster is represented by *minimal* tag(s) shared by *all* images in it. Due to the *concept-preserving* nature, the images in a cluster form an equivalence class with respect to the tags. Consequently, any image in each cluster can be selected as an exemplar without loss of accuracy to facilitate generation of high quality exemplar summary of the result set.

4. GEO-TEXTUAL DATA MANAGEMENT

The proliferation of GPS-equipped mobile devices has given rise to massive volumes of geo-textual or spatio-textual data (*e.g.,* points of interest, tweets, check-ins). Each geo-textual object is associated with a geo-location and a text value. In this section, we summarize the geo-textual data management challenges that we have addressed in our group.

4.1 Query Processing

A variety of classical spatial database queries and keyword queries have been revisited and rethought in the context of querying geo-textual data. Our research in this arena can be broadly classified into two streams: *spatial keyword queries* and *querying geo-textual streams*. In the former, we combine spatial functionality with keyword search (*e.g.,* find geo-tagged objects that best match the given location and keywords). Specifically, we have contributed to a variety of spatial keyword queries such as m-closest keywords [23], collective keyword [5], and keyword-aware route planning [4]. These queries typically find an aggregation of several geo-textual objects (ranked or otherwise) that are near each other. Some of our work have extended spatial queries on a spatial network that need to utilize spatial distance, which is more computationally expensive than Euclidean distance [6, 61, 62].

For the latter category, we focus on devising efficient solutions for querying streaming geo-tagged data (*e.g.,* microblog posts). Specifically, we have investigated techniques to support boolean subscription [8, 10] and similarity-based subscription [9] queries. These techniques aim to develop efficient spatial-keyword subscription strategies. More recently, we have looked into the problem of continuous queries on a stream of geo-tagged object (*e.g.,* detecting bursty region [22]).

4.2 Exploratory Search

We have also invented efficient techniques for exploring geo-tagged data. Our work can be broadly categorized into two streams: *region search* and *region exploration*. In the former, we aim to find a region for exploration that satisfies a user-defined condition (*e.g.,* size and shape of the region) and maximizes some aggregate score of the geo-tagged objects inside it [21]. In the latter category, we address the problem of exploring and discovering properties (*e.g.,* topics) of user-specified region [69].

5. INFORMATION PRIVACY

The era of big data has witnessed the collection, analysis, and sharing of individual data (*e.g.,* user behavioral records) at large scale. These data provide invaluable insights, but their usage often raises significant concerns about individual privacy. To address such concerns, a common practice is to anonymize the data by removing personal identifiers (such as names and IDs) and retaining all other information. This approach, however, has been shown to be vastly insufficient for privacy protection, since the information remained in the data may still be exploited to re-identify an individual. This motivated considerable research effort on systematic approaches for data privacy protection.

Our recent work on data privacy has focused on *differential privacy* [19], which is a strong and rigorous privacy model that has been adopted in Google Chrome and Apple iOS. In particular, we have de-

veloped techniques that improve the utility of differentially private algorithms for a number of important analytical tasks, including range count queries [56, 57, 64], model fitting [68], frequent itemset mining [50], histogram construction [55], and the synthesization of spatial, sequence, and high-dimensional data [30, 66, 67]. Most recently, we have investigated differentially private algorithms for collecting data from users who do not trust the data collector, and have devised solutions for collecting heavy hitters [41] and graph statistics [40]. In particular, our work in [56] was selected as one of the best papers in ICDE 2010.

6. FAIR PEER REVIEW MANAGEMENT

A fair peer-review process is a key ingredient for running a successful academic event. Fairness is affected by many factors, such as the expertise of reviewers, the quality of review comments, the design of the review form, etc. However, the most important factor is the relationships between authors and reviewers. In this research, we explore design and implementation of a novel reviewer suggestion system that focuses on declaration and detection of *conflicts of interest* (COIs) in the peer-review [53, 54], an issue that has received scant attention despite its significance in upholding quality and fairness of an academic event. This work is in collaboration with University of Macau and the Northeastern University, USA. Specifically, we extract relevant information related to authors by exploiting sources such as DBLP, *ResearchGate*, and *Arnet-Miner*. Next, we mine relationships between the authors based on various strategies such as meta-path information [45]. Finally, we rank the COIs and display a recommended COI list of a given set of authors by utilizing a supervised ranking model that can be iteratively refined from the data collected from past COI declarations. A prototype of our system called PISTIS[3] will be demonstrated in SIGMOD 2018 [54].

Acknowledgments. First of all, we thank our graduate students, research assistants, and research fellows for their hard work and contributions. We thank our current and past faculty members (Gao Cong, Arijit Khan, and James Cheng) for their vision and scientific contributions. Last but not the least we thank our international collaborators - these include Byron Choi, Graham Cormode, Curtis Dyreson, Johannes Gehrke, Wook-Shin Han, Christian S. Jensen, Divesh Srivastava, Ryan U, Marianne Winslett, Jeffrey Xu Yu, and Shuigeng Zhou.

[3]In Greek mythology, PISTIS was the personified spirit (*daimona*) of trust, honesty, and good faith.

7. REFERENCES

[1] S. S. Bhowmick, B.-S. Seah. Summarizing Biological Networks. *Springer Verlag*, May 2017.
[2] S. S. Bhowmick, A. Sun, B. Q. Truong. Why Not, WINE?: Towards Answering Why-Not Questions in Social Image Search. *In ACM MM*, 2013.
[3] S.S. Bhowmick, H.-E. Chua, B. Choi, C. Dyreson. `ViSual`: Simulation of Visual Subgraph Query Formulation to Enable Automated Performance Benchmarking. *IEEE TKDE* 29(8), 2017.
[4] X. Cao, L. Chen, G. Cong, X. Xiao. Keyword-aware Optimal Route Search. *In PVLDB*, 5(11), 2012.
[5] X. Cao, G. Cong, T. Guo, C. S. Jensen, B. C. Ooi. Efficient Processing of Spatial Group Keyword Queries. *ACM Trans. Database Syst.,* 40(2):13, 2015.
[6] X. Cao, G. Cong, et al. Retrieving Regions of Interest for User Exploration. *In PVLDB*, 7(9), 2014.
[7] A. Chapman and H. V. Jagadish. Why not?, *In SIGMOD*, 2009.
[8] L. Chen, G. Cong, X. Cao. An Efficient Query Indexing Mechanism for Filtering Geo-textual Data. *In SIGMOD*, 2013.
[9] L. Chen, G. Cong, et al. Temporal Spatial-keyword Top-k Publish/Subscribe. *In ICDE*, 2015.
[10] Z. Chen, G. Cong, Z. Zhang, T. Z. J. Fu, L. Chen. Distributed Publish/Subscribe Query Processing on the Spatio-Textual Data Stream. *In ICDE*, 2017.
[11] J. Cheng, Y. Ke, A. Fu, J. Xu Yu, L. Zhu. Finding Maximal Cliques in Massive Networks by H*-graph. *In SIGMOD*, 2010.
[12] J. Cheng, et al. Fast Algorithms for Maximal Clique Enumeration with Limited Memory. *In KDD*, 2012.
[13] J. Cheng, Y. Ke, S. Chu, M. T. Özsu. Efficient Core Decomposition in Massive Networks. *In ICDE*, 2011.
[14] J. Cheng, S. Huang, H. Wu, A. Fu. TF-Label: A Topological-folding Labeling scheme for Reachability Querying in a Large Graph. *In SIGMOD*, 2013.
[15] J. Cheng, Y. Ke, S. Chu, C. Cheng. Efficient Processing of Distance Queries in Large Graphs: A Vertex Cover Approach. *In SIGMOD*, 2012.
[16] J. Cheng, Y. Ke, et al. Fast Graph Query Processing with a Low-cost Index. *In VLDB J.* 20(4), 2011.
[17] Shumo Chu, James Cheng. Triangle listing in Massive Networks. *In TKDD*, 6(4), 2012.
[18] H. E. Chua, S.S. Bhowmick, L. Tucker-Kellogg. Synergistic Target Combination Prediction From Curated Signaling Networks: Machine Learning Meets Systems Biology and Pharmacology. *Methods*, Vol. 129, 2017.
[19] C. Dwork. Differential Privacy. *In ICALP*, 2006.
[20] K. Feng, G. Cong, S. S. Bhowmick, S. Ma. In Search of Influential Event Organizers in Online Social Networks. *In SIGMOD*, 2014.
[21] K. Feng, G. Cong, S. S. Bhowmick, W.-C. Peng, C. Miao. Towards Best Region Search for Data Exploration. *In SIGMOD*, 2016.
[22] K. Feng, T. Guo, G. Cong, S. S. Bhowmick, S. Ma. SURGE: Continuous Detection of Bursty Regions Over a Stream of Spatial Objects. *In ICDE*, 2018.
[23] T. Guo, X. Cao, G. Cong. Efficient Algorithms for Answering the m-Closest Keywords Query. *In SIGMOD*, 2015.
[24] T. Guo, X. Cao, G. Cong, J. Lu, X. Lin. Distributed Algorithms on Exact Personalized PageRank. *In SIGMOD*, 2017
[25] K. Huang, S. S. Bhowmick, S. Zhou, B. Choi. PICASSO: Exploratory Search of Connected Subgraph Substructures in Graph Databases. *PVLDB*, 10(12), 2017.
[26] H. H. Hung, S. S. Bhowmick, B. Q. Truong, B. Choi,

S. Zhou. QUBLE: Towards Blending Interactive Visual Subgraph Search Queries on Large Networks. *VLDB J.* 23(3), 2014.
[27] T. Ideker, T. Galitski, L. Hood. A New Approach to Decoding Life: Systems Biology. *Annual review of genomics and human genetics*, vol. 2, January 2001.
[28] C. Jin, S. S. Bhowmick, X. Xiao, J. Cheng, B. Choi. GBLENDER: Towards Blending Visual Query Formulation and Query Processing in Graph Databases. *In SIGMOD*, 2010.
[29] C. Jin, S. S. Bhowmick, et al. PRAGUE: A Practical Framework for Blending Visual Subgraph Query Formulation and Query Processing. *In ICDE*, 2012.
[30] G. Kellaris, S. Papadopoulos, X. Xiao, D. Papadias. Differentially Private Event Sequences over Infinite Streams. *In PVLDB*, 7(12), 2014.
[31] H. Kim, J. Lee, S. S. Bhowmick, W. S. Han, J. H. Lee, S. Ko, M. Jarrah. DUALSIM: Parallel Subgraph Enumeration in a Massive Graph on a Single Machine. *In SIGMOD*, 2016.
[32] X. Li, C. G. M. Snoek, M. Worring. Learning Social Tag Relevance by Neighbor Voting. *IEEE Transactions on Multimedia*, 11(7), 2009.
[33] H. Li, S. S. Bhowmick, et al. GETREAL: Towards Realistic Selection of Influence Maximization Strategies in Competitive Networks. *In SIGMOD*, 2015.
[34] H. Li, S. S. Bhowmick, A. Sun. CASINO: Towards Conformity-aware Social Influence Analysis in Online Social Networks. *In ACM CIKM*, 2011.
[35] H. Li, S. S. Bhowmick, A. Sun, J. Cui. Conformity-aware Influence Maximization in Online Social Networks. *In VLDB J*, 24(1), 2015.
[36] W. Lin, X. Xiao, G. Ghinita. Large-scale Frequent Subgraph Mining in MapReduce. *In ICDE*, 2014.
[37] W. Lin, X. Xiao, X. Xie, X. Li. Network Motif Discovery: A GPU Approach. *In ICDE*, 2015.
[38] Y. Liu, B. Zheng, X. He, Z. Wei, X. Xiao, K. Zheng, J. Lu. ProbeSim: Scalable Single-Source and Top-k SimRank Computations on Dynamic Graphs. *In PVLDB*, 11(1), 2017
[39] Y. Liu, J. Lu, H. Yang, X. Xiao, Z. Wei. Towards Maximum Independent Sets on Massive Graphs. *In PVLDB*, 8(13), 2015.
[40] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, K. Ren. Generating Synthetic Decentralized Social Graphs with Local Differential Privacy. *In CCS*, 2017.
[41] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, K. Ren. Heavy Hitter Estimation over Set-Valued Data with Local Differential Privacy. *In CCS*, 2016.
[42] B. S. Seah, S. S. Bhowmick, and A. Sun. PRISM: Concept-preserving social image search results summarization. *In ACM SIGIR*, 2014.
[43] B.-S. Seah, S. S. Bhowmick, et al. FUSE: Towards Multi-Level Functional Summarization of Protein Interaction Networks.*In BCB*, 2011.
[44] Y. Song, H. E. Chua, S. S. Bhowmick, B. Choi, S. Zhou. BOOMER: Blending Visual Formulation and Processing of p-Homomorphic Queries on Large Networks. *In ACM SIGMOD*, 2018.
[45] Y. Sun, J. Han, et al. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *In PVLDB*, 4(11), 2011.
[46] Y. Tang, X. Xiao, Y. Shi. Influence Maximization: Near-optimal Time Complexity meets Practical Efficiency. *In SIGMOD*, 2014.
[47] Y. Tang, Y. Shi, X. Xiao. Influence Maximization in Near-Linear Time: A Martingale Approach. *In SIGMOD*, 2015
[48] B. Tian, X. Xiao. SLING: A Near-Optimal Index Structure for SimRank. *In SIGMOD*, 2016.
[49] Jia Wang, James Cheng. Truss Decomposition in Massive Networks. *PVLDB* 5(9): 812-823, 2012.
[50] N. Wang, X. Xiao, et al. PrivSuper: A Superset-First Approach to Frequent Itemset Mining under Differential Privacy. *In ICDE*, 2017.
[51] S. Wang, R. Yang, X. Xiao, Z. Wei, Y. Yang. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. *In SIGKDD*, 2017
[52] S. Wang, Y. Tang, X. Xiao, Y. Yang, Z. Li. HubPPR: Effective Indexing for Approximate Personalized PageRank. *In PVLDB*, 2016.
[53] S. Wu, Leong H. U, S. S. Bhowmick, W. Gatterbauer. Conflict of Interest Declaration and Detection System in Heterogeneous Networks. *In CIKM*, 2017
[54] S. Wu, Leong H. U, S. S. Bhowmick, W. Gatterbauer. PISTIS: A Conflict of Interest Declaration and Detection System for Peer Review Management. *In SIGMOD*, 2018.
[55] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu. Differentially Private Histogram Publication. *In ICDE*, 2012.
[56] X. Xiao, G. Wang, J. Gehrke. Differential Privacy via Wavelet Transforms. *In ICDE*, 2010.
[57] X. Xiao, G. Bender, M. Hay, J. Gehrke. iReduct: Differential Privacy with Reduced Relative Errors. *In SIGMOD*, 2011.
[58] X. Xiao, Y. Tao, M. Chen. Optimal Random Perturbation at Multiple Privacy Levels. *PVLDB*, 2(1), 2009.
[59] M. Xie, S. S. Bhowmick, G. Cong, Q. Wang. PANDA: Towards Partial Topology-based Search on Large Networks in a Single Machine. In *VLDB J*, 26(2), 2016.
[60] Z. Xu, Y. Ke, Y. Wang, H. Cheng, J. Cheng. A Model-based approach to Attributed Graph Clustering. *In SIGMOD*, 2012.
[61] B. Yao, F. Li, X. Xiao. Secure Nearest Neighbor Revisited. *In ICDE*, 2013
[62] B. Yao, X. Xiao, F. Li, Y Wu. Dynamic Monitoring of Optimal Locations in Road Network Databases. *In VLDB J.*, 23(5), 2014.
[63] P. Yi, B. Choi, S. S. Bhowmick, J. Xu. AutoG: A Visual Query Autocompletion Framework for Graph Databases. *In The VLDB Journal*, 26(3), 2017.
[64] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, Z. Hao. Optimizing Batch Linear Queries under Exact and Approximate Differential Privacy. *ACM Trans. Database Syst.*, 40(2), 2015.
[65] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, X. Xiao. Private Release of Graph Statistics using Ladder Functions. *In SIGMOD*, 2015.
[66] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, X. Xiao. PrivBayes: Private Data Release via Bayesian Networks. *In TODS*, 42(4), 2017.
[67] J. Zhang, X. Xiao, X. Xie. PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions. *In SIGMOD*, 2016.
[68] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, M. Winslett. Functional Mechanism: Regression Analysis under Differential Privacy. *In PVLDB*, 5(11), 2012.
[69] K. Zhao, L. Chen, G. Cong. Topic Exploration in Spatio-temporal Document Collections. *In SIGMOD*, 2016.
[70] A. D. Zhu, W. Lin, S. Wang, X. Xiao. Reachability Queries on Large Dynamic Graphs: A Total Order Approach. *In SIGMOD*, 2014.
[71] A. D. Zhu, X. Xiao, S. Wang, W. Lin. Efficient Single-source Shortest Path and Distance Queries on Large Graphs. *In SIGKDD*, 2013.

Updates to the TODS Editorial Board

Christian S. Jensen
csj@cs.aau.dk

It is of paramount importance for a scholarly journal such as ACM Transactions on Database Systems to have a strong editorial board of respected, world-class scholars. The editorial board plays a fundamental role in attracting the best submissions, in ensuring insightful and timely handling of submissions, in maintaining the high scientific standards of the journal, and in maintaining the reputation of the journal. Indeed, the journal's Associate Editors, along with the reviewers and authors they work with, are the primary reason that TODS is a world-class journal.

As of January 1, 2018, five Associate Editors—Walid Aref, Graham Cormode, Gautam Das, Sabrina De Capitani di Vimercati, and Dirk Van Gucht—ended their terms, each having served on the editorial board for six years. In addition, they will stay on until they complete their current assignments.

Walid, Graham, Gautam, Sabrina, and Dirk have provided very substantial, high-caliber service to the journal and the database community. Specifically, they have lent their extensive experience, deep insight, and sound technical judgment to the journal. I have never seen them compromise on quality when handling submissions. Surely, they have had many other demands on their time, many of which are better paid, during these past six years. We are all fortunate that they have donated their time and unique expertise to the journal and our community during half a dozen years. They deserve our recognition for their commitment to the scientific enterprise.

Also as of January 1, 2018, five Associate Editors have joined the editorial board:

- Angela Bonifati, Université Claude Bernard Lyon 1
  `http://liris.cnrs.fr/angela.bonifati`
- Wolfgang Lehner, TU Dresden
  `https://wwwdb.inf.tu-dresden.de/our-group/team/wolfgang-lehner`
- Dan Olteanu, University of Oxford
  `http://www.cs.ox.ac.uk/dan.olteanu`
- Evaggelia Pitoura, University of Ioannina
  `http://www.cs.uoi.gr/~pitoura`
- Bernhard Seeger, University of Marburg
  `https://www.uni-marburg.de/fb12/arbeitsgruppen/dbs/team`

All five are highly regarded scholars in database systems. We are very fortunate that these outstanding scholars are willing to volunteer their valuable time and indispensable expertise for handling manuscripts for the benefit of our community. Indeed, I am gratified that they have committed to help TODS continue to evolve and improve, and I am looking forward to working with them.