Reformulating queries over web datasources

Michael Benedikt
Joint work with Balder ten Cate, Julien Leblay, Efi Tsamoyra
Outline

• Generating web query plans from proofs, by example.

• Brief tour of the theory of web query reformulation.

• Getting low-cost reformulations.

• The PDQ system: demo, experience, prospects.
Problem Statement

**Given:** a query Q and a schema describing a set of data sources – including integrity constraints relating sources, information about how sources can be accessed, and the cost of access.

**Goal:** generate a plan P having low cost that will completely answer Q

**Query:** conjunctive query – \( \exists x_1 \ldots x_m A_1(x_1 \ldots) \land \ldots \land A_m(x_m \ldots) \)

Equivalent to SQL basic SELECT or relational algebra Select/Project/Join query.

**Plan:** (informally) sequential program involving functions that access the sources. P completely answers Q: for each database instance I satisfying constraints \( P(I)=Q(I) \)

There may be many ways to answer Q. Or complete answers may be impossible.

The relations mentioned in Q may have multiple means of access: Web-service APIs, indexes, web forms.

Or they may have no access at all (e.g. virtual view, global schema in data integration).
Profinfo(eid, onum, lname) GetProfinfo: eid \rightarrow onum, lname
UDirectory(eid, lname) GetAll: {} \rightarrow eid, lname

Query Q = office number of professors named "Smith"

Unanswerable
Basic Example

Profinfo(eid, onum, lastname) GetProfinfo: eid \rightarrow onum, lname

UDirectory(eid, lname) GetAll: \{\} \rightarrow eid, lname

Inclusion Dependency (Referential Constraint):
Profinfo[eid, lname] \rightarrow UDirectory[eid, lname]

Query Q = office number of professors named "Smith"

Answerable, and there is an obvious plan.
Subsumes Many DB Problems

Querying on top of web forms or web services
Querying using materialized views
Querying in data integration
Optimizing queries using constraints
Cost-based simplification of complex queries
Querying on top of Web forms/Web Services

FindApt: Region, NumRooms, Area → AptName, Address, Price
Notion of **binding patterns** introduced by Ullman in 1997

**Original Motivation:** we have a fixed set of key-looked up indices

We want to know if a query $Q$ can be answered using only the indices

Related to the goal of reformulating a query over a set of "physical structures"
Some History

First theoretical work on querying with binding patterns by Chang and Li [CL2001TODS] – complexity of determining if a query can be answered in the absence of constraints.

Florescu, Levy, Manolescu, Suciu {FLMS99} Study of impact of access patterns on rule-based optimizeion
Some History

Problem studied intensively by Alan Nash 1965-2009

Several papers on querying with access patterns, both on his own and with Bertram Ludascher and Alin Deutsch

Cali and Martinenghi analysis of containment and optimization problems
Cali and Martinenghi analysis of containment and optimization problems.

Related work on containment problem by B., Bourhis, Gottlob, Senellart.
**Goal:** reformulate a query $Q$ over target class $T$ with respect to integrity constraints $\Sigma$

**General Methodology**

- Pick out a property that the query $Q$ should have in order to be answerable with respect to access patterns and constraints
- Write out the property as an implication/proof goal in logic
- Look for a proof witnessing this implication
- Apply an **interpolation algorithm** to extract a reformulation from the proof
**Goal**: reformulate a query $Q$ over target class $T$ with respect to integrity constraints $\Sigma$

**General Methodology**

- Pick out a property that the query $Q$ should have in order to be answerable with respect to access patterns and constraints
- Write out the property as an implication/proof goal in logic

We'll need "axioms" that capture reasoning with access methods. We will focus on the axioms, not the properties they capture

- Look for a proof witnessing this implication
- Apply an **interpolation algorithm** to extract a reformulation from the proof
Add to schema:

• relation **Accessible(x)**: "value x can be exposed in an access"
• for each relation **R(x₁…xₙ)** in the schema, add an "inferred accessible version" **InfAccR(x₁…xₙ)**: "fact **R(x₁…xₙ)** knowable via accesses + reasoning"

• new integrity constraints "accessibility axioms", capturing semantics of accesses.
New integrity constraints, capturing semantics of accesses. In the example:

∀ eid ∀ Iname UDirectory(eid, Iname) →
[InfAccUDirectory(eid, Iname) ∧ Accessible(eid) ∧ Accessible(Iname)]  /* access UDirectory */

∀ eid ∀ onum ∀ Iname
Accessible(eid) ∧ Profinfo(eid, onum, Iname) →
[InfAccProfinfo(eid, onum, Iname) ∧ Accessible(onum) ∧ Accessible(Iname)]  /* access Profinfo on eid */
Reasoning about Access

Accessible Schema (AccSch) =

- Original integrity constraints (on original relation names) +
- Accessibility Axioms:

For each relation $R$ of arity $n$ with an access method having input on the first $m$ positions have accessibility axiom:

$$\forall x_1 \ldots x_n \text{Accessible}(x_1) \land \ldots \land \text{Accessible}(x_m) \land R(x_1 \ldots x_n) \rightarrow$$

$$[$$

[InfAcc(R)(x_1 \ldots x_n) \land \land_{i \geq m} \text{Accessible}(x_i)$]

- Integrity constraints on $\text{InfAcc}$ relations
Reasoning about Answerability

What can one do with these rules?

Try to show (using these axioms) that:
if \( Q \) holds in a database, then a user would know that \( Q \) holds

Assume \( Q: \exists \text{eid} \ \text{ProfInfo}(\text{eid}, \text{onum}, "\text{Smith}"")

Prove: \( \text{InfAcc}(Q) = \exists \text{eid} \ \text{InfAccProfinfo}(\text{eid}, \text{onum}, "\text{Smith}"") \)

Given query \( Q \), "inferred accessible copy of \( Q \)" = \( \text{InfAcc}(Q) \)
= "\( Q \) is knowable via accesses and inference"

Change all relations in \( Q \) to their inferred accessible versions.
"Prove \( \text{InfAcc}(Q) \) from \( Q \) using axioms" means
Prove \( \forall \ x_1, ..., x_n \ Q(x_1 ... x_n) \land \text{Axioms} \rightarrow \text{InfAcc}(Q)(x_1 ... x_n) \)
To show that $Q$ is answerable, show that $Q$ implies $\text{InfAcc}(Q)$ w.r.t the accessible schema (write $Q \vdash \text{InfAcc}(Q)$ wrt $\text{AccSch}$).

Proof

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

$\text{Profinfo}(e_0, \text{onum}, "\text{Smith"})$

$\text{UDirectory}(e_0, "\text{Smith"}) \quad /* \text{Integrity Constraint} */$

$\text{InfAccUDirectory}(e_0, "\text{Smith"}) \quad /* \text{Accessibility Axiom} */$

$\text{Accessible}(e_0) \quad /* \text{Accessibility Axiom} */$

$\text{InfAccProfinfo}(e_0, \text{onum}, "\text{Smith"}) \quad /* \text{Accessibility Axiom} */$

$\text{InfAcc}(Q) = \exists \, \text{eid} \, \text{InfAccProfinfo}(\text{eid}, \text{onum}, "\text{Smith"})$
To show that $Q$ is answerable, we can show that $Q \vdash \text{InfAcc}(Q)$ wrt $\text{AccSch}$

**Proof**

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

$\text{Profinfo}(e_0, \text{onum, "Smith"})$

$\text{UDirectory}(e_0, \text{"Smith"})$

$\text{InfAccUDirectory}(e_0, \text{"Smith"})$

$\text{Accessible}(e_0)$

$\text{InfAccProfinfo}(e_0, \text{onum, "Smith"})$

$\text{InfAcc}(Q) = \exists \text{eid } \text{InfAccProfinfo}(\text{eid, onum, "Smith"})$

**Plan**

Access $\text{UDirectory}$ with no input, putting the tuples with $\text{iname} =$ "Smith" in a table $T$ of $e_0$'s

Access $\text{Profinfo}$ with input $T$ joined with "Smith", put projection of resulting tuples in table $T_2$ with attribute $\text{onum}$

Output $T_2$
Proof to Plan, Formally

To show that \( Q \) is answerable, we can show that \( Q \vdash \text{InfAcc}(Q) \) wrt \( \text{AccSch} \)

Proof

\[
Q = \exists \text{eid} \ \text{ProfInfo(eid, onum, "Smith")}
\]

\[
\text{Profinfo(e}_0\text{, onum, "Smith")}
\]

\[
\text{UDirectory(e}_0\text{, "Smith")}
\]

\[
\text{InfAccUDirectory(e}_0\text{, "Smith")}
\]

\[
\text{Accessible(e}_0\text{)}
\]

\[
\text{InfAccProfinfo(e}_0\text{, onum, "Smith")}
\]

\[
\text{Accessible(onum)}
\]

\[
\text{InfAccProfinfo(e}_0\text{, onum, "Smith")}
\]

\[
\text{InfAcc}(Q) = \exists \text{eid} \ \text{InfAccProfinfo(eid, onum, "Smith")}
\]

We will now state some theorems of the form:

*If*

\( Q \vdash \text{InfAcc}(Q) \)

*Then*

we can get a plan that completely answers \( Q \) (and vice versa)
Reasoning with Access Patterns

The "positive" derived schema \textbf{AccSch} has two copies of each relation, \textbf{R}, \textbf{InfAccR}, along with relation \textbf{accessible}, and axioms going from \textbf{R} to \textbf{InfAccR} and \textbf{accessible}.

The "bi-directional version" \textbf{AccSch} \leftrightarrow treats \textbf{InfAccR} and \textbf{R} symmetrically. We also have rules in the opposite direction.

For each relation \textbf{R} of arity \textbf{n} with an access method having input the first \textbf{m} positions, include an axiom:

\[ \forall x_1\ldots x_n \quad \text{Accessible}(x_1) \land \ldots \land \text{Accessible}(x_m) \land \text{InfAccR}(x_1\ldots x_n) \rightarrow \]
\[ [ \text{R}(x_1\ldots x_n) \land \bigwedge_{i \geq m} \text{Accessible}(x_i) ] \]
First Main Theorem

**Theorem:** If constraints are in first-order logic, then $Q$ has an RA plan iff $Q \vdash \text{InfAcc}(Q)$ wrt $\text{AccSch}$

**RA plan:** sequence of commands that are either:

- **access commands:** $T' \leftarrow \text{mt} \leftarrow T$
  where $\text{mt}$ is a method of the schema, $T$ is a temporary table that matches the input of $\text{mt}$, $T'$ a temporary table matching the output of $\text{mt}$.

- **middleware query commands:** $T = \mathcal{E}(T_1 \ldots T_n)$,
  where $T, T_1 \ldots T_n$ temporary tables, $\mathcal{E}$ in relational algebra

Plans have a distinguished output table.
Theorem: If constraints are in first-order logic, then Q has a USPJ plan iff $Q \vdash \text{InfAcc}(Q)$ wrt AccSch.

USPJ plan: as before, but middleware query commands are $T = E(T_1 \ldots T_n)$, $T$, $T_1 \ldots T_n$ temporary tables, $E$ a USPJ query.

Version for "positive existential plans".
The additional "backward axioms" capture "negative information" revealed by an access.

E.g. if $R$ has an input-free access then we have rule:
\[
\forall x_1 \ldots x_n \neg R(x_1 \ldots x_n) \rightarrow \neg \text{InfAccR}(x_1 \ldots x_n)
\]

**Informally:**
if a certain lookup on $R$ really has no matches, then a user will know that it has no matches.
Equivalence Theorems

Reformulation Goal

- \( Q \) has an RA-plan using the access methods equivalent w.r.t the constraints
- \( Q \) has a USPJ-plans using the access methods equiv w.r.t constraints

Provability property

- \( Q \vdash \text{InfAcc}(Q) \) via \( \text{AccSch} \)
- \( Q \vdash \text{InfAcc}(Q) \) via \( \text{AccSch} \)
Equivalence Theorems

Reformulation Goal

\( Q \) has an RA-plan using the access methods equivalent w.r.t the constraints

\( Q \) has a USPJ-plan using the access methods equivalent w.r.t constraints

\( Q \) has a USPJ\(^\neg\) plan equivalent w.r.t. constraints

Provability property

\( Q \vdash \text{InfAcc}(Q) \) via \( \text{AccSch} \)

\( Q \vdash \text{InfAcc}(Q) \) via another version of the axioms \( \text{AccSch}\neg \)
<table>
<thead>
<tr>
<th>Reformulation Goal</th>
<th>Semantic Property</th>
<th>Provability property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q has an RA-plan using the access methods equivalent w.r.t the constraints</td>
<td>Q is determined by the accessible facts w.r.t the constraints</td>
<td>Q ⊨ InfAcc(Q) via AccSch$\leftrightarrow$</td>
</tr>
<tr>
<td>Q has a USPJ-plans using the access methods equiv w.r.t constraints</td>
<td>Q is monotone in the accessible data w.r.t. the constraints</td>
<td>Q ⊨ InfAcc(Q) via AccSch</td>
</tr>
<tr>
<td>Q has a USPJ$^\neg$ plan equivalent w.r.t. constraints</td>
<td>Q is induced-subinstance monotone in the accessible data of the data w.r.t. constraints</td>
<td>Q ⊨ InfAcc(Q) via AccSch$\neg$</td>
</tr>
</tbody>
</table>

See BCTPODS14
Theorem:
If integrity constraints are in first-order logic, then

Q has an RA plan iff $Q \vdash \text{InfAcc}(Q)$ using AccSch$\leftrightarrow$

Q has a USPJ plan iff $Q \vdash \text{InfAcc}(Q)$ using AccSch

Proof Idea: Given a proof that witnesses $Q \vdash \text{InfAcc}(Q)$, apply an interpolation algorithm.

We prove a new interpolation result, "Access Interpolation", that shows that the interpolation method gives us the reformulation we want.
Recall

Example

Profinfo(eid, onum, lastname)  GetProfinfo: eid \rightarrow onum, lname

UDirectory(eid, lname)        GetAll: {} \rightarrow eid, lname

Inclusion Dependency (Referential Constraint):
Profinfo[eid,lname] \rightarrow UDirectory[eid,lname]

Query $Q =$office number of professors named "Smith"

Answerable, via an algorithm that used chase proofs.
Recall Example

Proof

\[ Q = \exists \text{ eid ProfInfo(eid, onum, "Smith")} \]

Profinfo(e_0, onum, "Smith")
UDirectory(e_0, "Smith")
AccessedUDirectory(e_0, "Smith")
Accessible(e_0)
AccessedProfinfo(e_0, onum, "Smith")
Accessible(onum)

InfAccProfinfo(e_0, onum, "Smith")
InfAcc(Q) = Accessible(onum) \land 
\exists \text{ eid InfAccProfinfo(eid, onum, "Smith")}
Chase algorithm

To see if $Q$ implies $Q'$ with respect to TGDs $\Gamma$

"Chase Algorithm" to decide if $Q$ implies $Q'$ with respect to constraints $\Gamma"
Chase algorithm

$Q = \exists x \ y \ T(x, y) \land R(x, y)$

$Q' = \exists x \ z \ T(x, y) \land S(x, z)$

$I' = \forall x \ y \ R(x, y) \rightarrow \exists z \ S(x, z)$

"Chase Algorithm" to decide if $Q$ implies $Q'$ with respect to
constraints $I'$

$Q = \exists x \ y \ T(x, y) \land R(x, y)$

$T(x_0, y_0) \land R(x_0, y_0)$

Chase step: Add $S(x_0, z_0)$

$Q'$ holds

So $Q$ does imply $Q'$ w.r.t. constraints
Chase-based Proof

Proof

\[ Q = \exists \text{eid} \text{ProfInfo}(\text{eid}, \text{onum}, "Smith") \]
\[ \text{Profinfo}(e_0, \text{onum}, "Smith") \]

UDirectory(e_0, "Smith")

\[ \text{AccessedUDirectory}(e_0, "Smith") \]

Accessible(e_0)

\[ \text{AccessedProfinfo}(e_0, \text{onum}, "Smith") \]

Accessible(onum)

\[ \text{InfAccProfinfo}(e_0, \text{onum}, "Smith") \]

\[ \text{InfAcc}(Q) = \text{Accessible}(\text{onum}) \land \exists \text{eid} \text{InfAccProfinfo}(\text{eid}, \text{onum}, "Smith") \]

Plan

Access UDirectory with no input, putting the tuples with \text{name}="Smith" in a table \( T_1 \) of \( e_0 \)'s

Access \text{Profinfo} with input \( T_1 \) joined with "Smith", put projection of resulting tuples in table \( T_2 \) with attribute \text{onum}

Output \( T_2 \)
Chase-based Plans

When $j^{th}$ accessibility axiom fires, the plan stores a table $T_j$ having attributes for every chase constant that is accessible after step $j$ of the proof.

At run time tuples stored in table $T_j$ with attributes $c_0, c_1, c_2 \ldots$ are all the candidate tuples that "could agree with $c_0, c_1, c_2 \ldots$" That is: all tuples that have homomorphisms from the $j^{th}$ configuration.

Proof State

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

Profinfo($e_0$, onum, "Smith")

UDirectory($e_0$, "Smith")

AccessedUDirectory($e_0$, "Smith")

Accessible($e_0$)

AccessedProfinfo($e_0$, onum, "Smith")

Accessible(onum)

InfAccProfinfo($e_0$, onum, "Smith")

InfAcc($Q$) $= \text{Accessible(onum) } \land \exists \text{eid InfAccProfinfo}(\text{eid, onum, "Smith"})$

Plan Actions

Access $\text{UDirectory}$ with no input putting result in a table $T_1$ of $e_0$'s

Access $\text{Profinfo}$ with input $T_1$ joined with "Smith", put projection of resulting tuples in table $T_2$

Output $T_2$
We have a chase-based algorithm, and we have the algorithm that comes from interpolation.

Which is better?

They are the same.
Chase-based Plans

Still need to find the proof, and if there is no proof, chase may go on forever—no bound on how long it can take.

• Many subclasses where the chase terminates — after a certain number of steps, no new rules can fire. E.g. Acyclic Referential Constraints

• Many subclasses where the chase does not terminate, but we can truncate it after a while knowing that we will not miss a proof. E.g. Guarded TGDs.

• For general TGDs, we have no bound, but we can simply stop after a certain point.
Plans and Cost

\[\text{Profinfo}(\text{eid, onum, lastname}) \text{ GetProfinfo: } \text{eid} \rightarrow \text{onum, lname} \]

\[\text{UDirect}_1(\text{eid, lname}), \text{ GetAll}_1: \{\} \rightarrow \text{eid, lname} \]
\[\text{UDirect}_2(\text{eid, lname}), \text{ GetAll}_2: \{\} \rightarrow \text{eid, lname} \]
\[\text{UDirect}_3(\text{eid, lname}) \text{ GetAll}_3: \{\} \rightarrow \text{eid, lname} \]

Different costs of access to each \text{UDirect}_i, different selectivity within \text{Profinfo}

Inclusion Dependency: \text{Profinfo}[\text{eid, lname}] \rightarrow \text{UDirect}_1[\text{eid, lname}]
\rightarrow \text{UDirect}_2[\text{eid, lname}]
\rightarrow \text{UDirect}_3[\text{eid, lname}]

How many plans?
At least 7 plans. And which is better depends on the cost and selectivity.
Plans and Cost

Profinfo(eid, onum, lastname) GetProfinfo: eid \(\rightarrow\) onum,lname

UDirect_1(eid, lname), GetAll_1: {} \(\rightarrow\) eid, lname
UDirect_2(eid, lname), GetAll_2: {} \(\rightarrow\) eid, lname
UDirect_3(eid,lname) GetAll_3: {} \(\rightarrow\) eid, lname

Different costs of access to each UDirect_i, different selectivity within Profinfo

Inclusion Dependency: Profinfo[eid,lname] \(\rightarrow\) UDirect_1[eid,lname]
\(\rightarrow\) UDirect_2[eid, lname]
\(\rightarrow\) UDirect_3[eid, lname]

Moral: for every interesting plan, there is a corresponding proof. But cost is not reflected in the proof structure.
Cost-based Proof/Plan Search

Two settings:

Simple cost function: one that associates each access method with a cost; cost of plan is sum of cost of methods called.

Black-box cost function: any function on plans that is monotone as accesses grow.

Approach:

• Explore the space of all proofs.
• While exploring, generate partial plan and estimate cost.
Proof/Plan Search

Assumption Q

Proof config: Match for InfAcc(Q)

Calculate Cost

Calc Cost

Fire some initial Integrity Constraint rules

Fire Acc. Axiom/Access UDirect_1

Access UDirect_3

Access UDirect_2

Access UDirect_2

Access ProfInfo

Proof Config.

Proof Config.
Proof-driven planning

Generate tree of possible sequences of accesses.

With every node, keep track of the corresponding proof.

If a node has a complete proof, we know the plan is successful – terminate that branch, backtrack to look for better plans.
System **PDQ (Proof-driven querying)** for searching proof/plan space.

- Proof-related search optimizations – direct search via proof structure.
- Cost-related search optimizations – direct search by properties of cost function.

Flavor of the features in next few slides.
Proof-related Optimization

**Problem:** Many integrity constraint rules can be fired – often infinitely many.

Fire "all" cost-free rules after every access rule.

For common classes of constraints, we can cut off the number of rules that could lead to something useful for a proof.

Currently focus on guarded TGDs, where we use the "guarded blocking condition".
**Problem:** Can fire any number of access rules.

Access rules are costly: truncate at a certain depth.

Prune partial plans in the tree if there is another one with lower cost and which has generated "more" facts.
Lots of interleavings of accesses are the same.

Detect when corresponding proof states (chase configurations) are "similar"

Similar = "similar chase configuration"
Proof/Plan Search

Assumption Q

- Fire some initial Integrity Constraint rules
- Calculate Cost
- Match for InfAcc(Q)
- Identify similar configurations.

Turns plan space into a DAG – need to account for DAG structure during subsequent exploration.
Dynamic Programming

For simple cost functions, maintain **best cost of path to success** at each node, and update (propagating up DAG) whenever a successful match is found.

For blackbox cost functions, maintain all **paths to success** at each node, and update when match is found.

Key new twist over generic search is detecting similarity of states. Generally requires an isomorphism check – expensive.
Use a stricter check by requiring more of a match.
PDQ demo

See http://www.cs.ox.ac.uk/pdq/
Discussion

Bottlenecks

• Runtime
  • Access cost
  • Need for bushy trees/bushy proofs

• Planning time issues:
  • Chasing
  • Heuristics for exploration

• Interface issues
  • Constraint specification/discovery
  • Cost estimation
Conclusions

- Close correspondence between proofs and query plans.
  - General approach via interpolation that allows you to read off plans from proofs

- Many degrees of freedom
  - Tweaking the axioms, you can control the form of the plans (RA-plans, USPJ-plans, etc.)
  - Different proof systems (e.g. chase, saturation-based, mosaics ...)
  - Different interpolation algorithms
  - Different search strategies

- Applicable to database problems where it pays to search a wide space of plans before runtime