Query Optimization and Compilation

David Toman

D.R. Cheriton School of Computer Science
University of Waterloo

Joint work with Alexander Hudek and Grant Weddell

Introduction: Review&Goal

STANDARD APPROACH(es)

- Physical Data Independence
- Relational Algebra and Access Paths
- Relational Algebra Equivalences and Plans
- Virtual Machine executing Plans

Introduction: Review&Goal

STANDARD APPROACH(es)

- Physical Data Independence
- Relational Algebra and Access Paths
- Relational Algebra Equivalences and Plans
- Virtual Machine executing Plans

ALTERNATIVE

ALL rewriting/optimization in Relational Calculus (+Schema Constraints)
followed by a *simple* Code Generator (+Library of Access Path Templates)

Introduction: Physical Data Independence

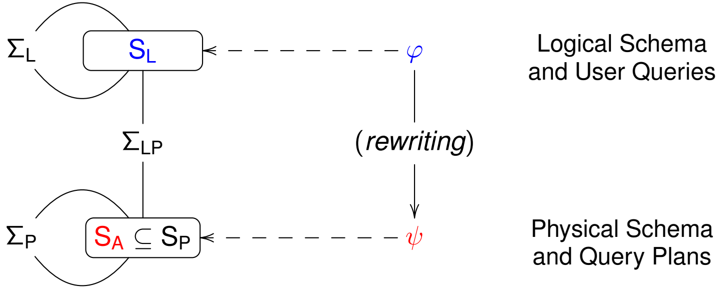
Definability and Rewriting

Queries	range-restricted FOL (a.k.a. SQL)
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma^L \cup \Sigma^{LP} \cup \Sigma^P$
Data	CWA (complete information)

Introduction: Physical Data Independence

Definability and Rewriting

Queries	range-restricted FOL over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma^L \cup \Sigma^{LP} \cup \Sigma^P$
Data	CWA (complete information for S_A symbols)



[Borgida, de Bruijn, Franconi, Seylan, Straccia, Toman, Weddell: On Finding Query Rewritings under Expressive Constraints. SEBD 2010: 426-437]

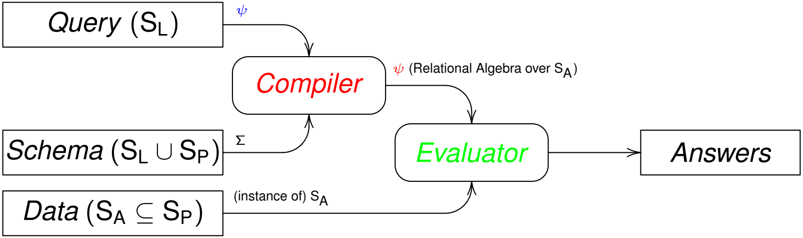
Introduction: Physical Data Independence

Definability and Rewriting

Queries	range-restricted FOL	over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL	$\Sigma := \Sigma^L \cup \Sigma^{LP} \cup \Sigma^P$
Data	CWA (complete information	for S_A symbols)

- to users it looks like a *single model* (of the logical schema)
- implementation can pick from many models

but *definable* queries answer the same in each of them



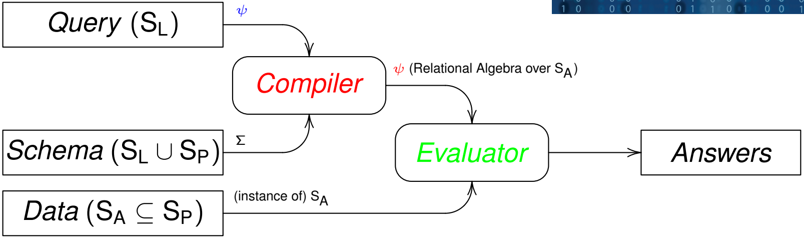
Introduction: Physical Data Indepen

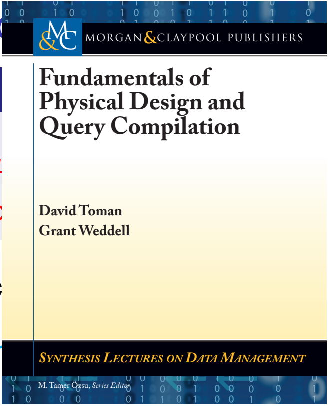
Definability and Rewriting

Queries	range-restricted FOL over S_L
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma^L$
Data	CWA (complete information fo

- to users it looks like a *single model* (of the logic
- implementation can pick from many models

but *definable* queries answer





©2011

What can this do?

GOAL

Generate query plans *that compete with hand-written programs in C*

1. standard designs and materialized viwes,
2. linked data structures, pointers, . . .
3. hash-based access to data (including hash-joins),
4. multi-level storage (aka disk/remote/distributed files), . . .

. . . all without having to code (too much) in C/C++ !

What can this do: Standard Designs

Base File

$$\texttt{table}(x, y, z) \leftrightarrow \exists r.\texttt{basefile}(r, x, y, z)$$
$$\texttt{basefile}(r, x_1, y_1, z_1) \wedge \texttt{basefile}(r, x_2, y_2, z_2) \rightarrow (x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2)$$

Indices (and index-only plans)

$$\texttt{index1}(x, r) \leftrightarrow \exists y, z.\texttt{basefile}(r, x, y, z)$$
$$\texttt{index2}(y, r) \leftrightarrow \exists x, z.\texttt{basefile}(r, x, y, z)$$
$$\texttt{index3}(z, r) \leftrightarrow \exists y, z.\texttt{basefile}(r, x, y, z)$$

Horizontal Partitioning and Inheritance

$$(\texttt{hp1}(r, x, y, z) \vee \texttt{hp2}(r, x, y, z)) \leftrightarrow \exists r.\texttt{basefile}(r, x, y, z)$$
$$(\texttt{hp1}(r, x, y, z) \wedge \texttt{hp2}(r, x, y, z)) \rightarrow \texttt{false}$$

...

What can this do: Standard Designs

Base File

$$\texttt{table}(x, y, z) \leftrightarrow \exists r.\texttt{basefile}(r, x, y, z)$$
$$\texttt{basefile}(r, x_1, y_1, z_1) \wedge \texttt{basefile}(r, x_2, y_2, z_2) \rightarrow (x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2)$$

Indices (and index-only plans)

$$\texttt{index1}(x, r) \leftrightarrow \exists y, z.\texttt{basefile}(r, x, y, z)$$
$$\texttt{index2}(y, r) \leftrightarrow \exists x, z.\texttt{basefile}(r, x, y, z)$$
$$\texttt{index3}(z, r) \leftrightarrow \exists y, z.\texttt{basefile}(r, x, y, z)$$

Horizontal Partitioning and Inheritance

$$(\texttt{hp1}(r, x, y, z) \vee \texttt{hp2}(r, x, y, z)) \leftrightarrow \exists r.\texttt{basefile}(r, x, y, z)$$
$$(\texttt{hp1}(r, x, y, z) \wedge \texttt{hp2}(r, x, y, z)) \rightarrow \texttt{false}$$

...

...physical design captured via (i) physical tables and (ii) constraints (and (iii) a cost model)

What can this do: Materialized Views

Puzzle

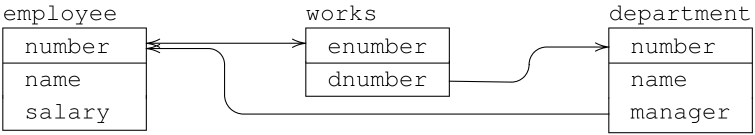
Views:
$$V_1(x, y) \leftrightarrow \exists t, u, v.R(t, x) \wedge R(t, u) \wedge R(u, y)$$
$$V_2(x, y) \leftrightarrow \exists u.R(x, u) \wedge R(u, y)$$
$$V_3(x, y) \leftrightarrow \exists t, u.R(x, t) \wedge R(t, u) \wedge R(u, y)$$

Query:
$$Q(x, y) \leftrightarrow \exists t, u, v.R(t, x) \wedge R(t, u) \wedge R(u, v) \wedge R(v, y)$$

CAN Q BE EXPRESSED USING THE MATERIALIZED VIEWS V_1, V_2, AND V_3?

Lists and Pointers

1. Logical Schema



2. Physical Design: a *linked list of* `emp` *records pointing to* `dept` *records*.

```
record emp of                        record dept of
       integer    num                       integer    num
       string     name                      string     name
       integer    salary                    reference  manager
       reference  dept
```

3. Access Paths: `empfile`/1/0, `emp-num`/2/1, ... (but no `deptfile`)

4. Integrity Constraints (many), e.g.,

$$\forall x, y, z.\mathtt{employee}(x, y, z) \rightarrow \exists w.\mathtt{empfile}(w) \wedge \mathtt{emp\text{-}num}(w, x),$$
$$\forall a, x.\mathtt{empfile}(a) \wedge \mathtt{emp\text{-}num}(a, x) \rightarrow \exists y, z.\mathtt{employee}(x, y, z), \ldots$$

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

or, in C-like syntax:

```
for a in empfile do
    x := a->num;
    y := a->name;
```

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

$$\exists a.\texttt{empfile}(a) \land \texttt{emp-num}(a, x) \land \texttt{emp-name}(a, y)$$

2. List all department numbers with their manager names ($\exists z, u, v.\texttt{department}(x, z, u) \land \texttt{employee}(u, y, v)$):

$$\land\ \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, e) \land \texttt{emp-name}(e, y)$$

$$\land\ \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, b) \land \texttt{compare}(a, b)$$

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

   $$\exists a.\texttt{empfile}(a) \land \texttt{emp-num}(a, x) \land \texttt{emp-name}(a, y)$$

2. List all department numbers with their manager names
   ($\exists z, u, v.\texttt{department}(x, z, u) \land \texttt{employee}(u, y, v)$):

   $$\exists a, d, e.\texttt{empfile}(a) \land \texttt{emp-dept}(a, d)$$
   $$\land\ \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, e) \land \texttt{emp-name}(e, y)$$

   $\Rightarrow$ needs “departments have at least one employee”.

   $$\land\ \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, b) \land \texttt{compare}(a, b)$$

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

   $$\exists a.\texttt{empfile}(a) \land \texttt{emp-num}(a, x) \land \texttt{emp-name}(a, y)$$

2. List all department numbers with their manager names ($\exists z, u, v.\texttt{department}(x, z, u) \land \texttt{employee}(u, y, v)$):

   $$\exists a, d, e.\texttt{empfile}(a) \land \texttt{emp-dept}(a, d) \land \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, e) \land \texttt{emp-name}(e, y)$$

   $\Rightarrow$ needs “departments have at least one employee”.

   $$\exists a, b, d.\texttt{empfile}(a) \land \texttt{emp-name}(a, y) \land \texttt{emp-dept}(a, d) \land \texttt{dept-num}(d, x) \land \texttt{dept-mgr}(d, b) \land \texttt{compare}(a, b)$$

   $\Rightarrow$ needs “managers work in their own departments”.

What can this do: navigating pointers

1. List all employee numbers and names ($\exists z, w.\texttt{employee}(x, y, z, w)$):

$$\exists a.\texttt{empfile}(a) \wedge \texttt{emp-num}(a, x) \wedge \texttt{emp-name}(a, y)$$

2. List all department numbers with their manager names ($\exists z, u, v.\texttt{department}(x, z, u) \wedge \texttt{employee}(u, y, v)$):

$$\exists a, d, e.\texttt{empfile}(a) \wedge \texttt{emp-dept}(a, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-mgr}(d, e) \wedge \texttt{emp-name}(e, y)$$

$\Rightarrow$ needs "departments have at least one employee".

…needs *duplicate elimination* during projection.

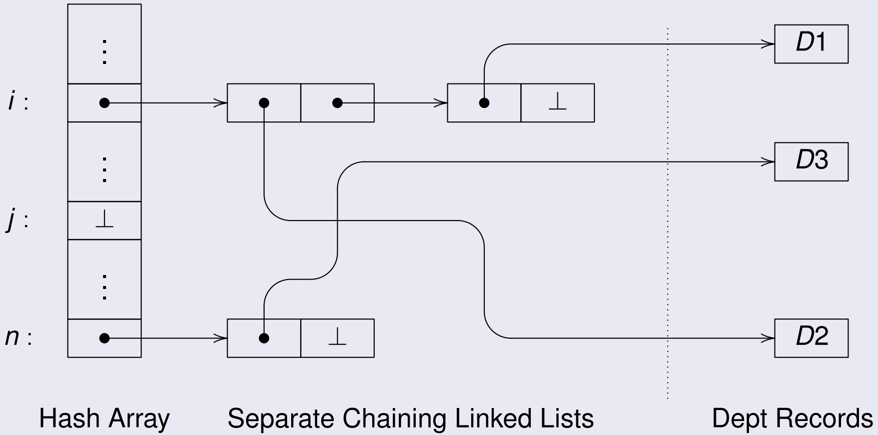
$$\exists a, b, d.\texttt{empfile}(a) \wedge \texttt{emp-name}(a, y) \wedge \texttt{emp-dept}(a, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-mgr}(d, b) \wedge \texttt{compare}(a, b)$$

$\Rightarrow$ needs "managers work in their own departments".

…NO *duplicate elimination* during projection.

What can it do: Hashing, Lists, et al.

Hash Index with (list-based) Separate Chaining



What can it do: Hashing, Lists, et al.

Hash Index on `department`'s `name`:

Access paths:

$S_A \supseteq \{\texttt{hash}/2/1, \texttt{hasharraylookup}/2/1, \texttt{listscan}/2/1\}$.

Physical Constraints:

$$\begin{aligned}\Sigma_{LP} \supseteq \{ & \forall x,y.((\texttt{deptfile}(x) \wedge \texttt{dept-name}(x,y)) \rightarrow \exists z,w.(\texttt{hash}(y,z) \\ & \qquad \wedge \texttt{hasharraylookup}(z,w) \wedge \texttt{listscan}(w,x))), \\ & \forall x,y.(\texttt{hash}(x,y) \rightarrow \exists z.\texttt{hasharraylookup}(y,z)), \\ & \forall x,y.(\texttt{listscan}(x,y) \rightarrow \texttt{deptfile}(y)) \qquad \}\end{aligned}$$

What can it do: Hashing, Lists, et al.

Hash Index on `department`'s `name`:

Access paths:

$S_A \supseteq \{\texttt{hash}/2/1, \texttt{hasharraylookup}/2/1, \texttt{listscan}/2/1\}$.

Physical Constraints:

$$\Sigma_{LP} \supseteq \{\forall x, y.((\texttt{deptfile}(x) \wedge \texttt{dept-name}(x, y)) \rightarrow \exists z, w.(\texttt{hash}(y, z) \wedge \texttt{hasharraylookup}(z, w) \wedge \texttt{listscan}(w, x))),$$
$$\forall x, y.(\texttt{hash}(x, y) \rightarrow \exists z.\texttt{hasharraylookup}(y, z)),$$
$$\forall x, y.(\texttt{listscan}(x, y) \rightarrow \texttt{deptfile}(y)) \quad \}$$

Query:

$\exists y, z.(\texttt{department}(x_1, p, y) \wedge \texttt{employee}(y, x_2, z))\{p\}$.

```
E?x6.(Hash(p,?x6)^E?x5.(Hasharraylookup(?x6,?x5)
                   ^E?x4.(Listscan(?x5,?x4)
                   ^E?s0.(Dept-name(?x4,?s0)^Cmp(p,?s0))
                   ^Dept-num(?x4,x1)
                   ^E?x3.(Dept-manager(?x4,?x3)^Emp-name(?x3,x2))))
```

What can this do: two-level store

The access path `empfile` is refined by `emppages`/1/0 and `emprecords`/2/1:

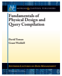
`emppages` returns (sequentially) disk pages containing `emp` records, and

`emprecords` given a disc page, returns `emp` records in that page.

5 List all employees with the same name
$(\exists z, u, v, w, t.\texttt{employee}(x_1, z, u, v) \wedge \texttt{employee}(x_2, z, w, t))$:

$$\exists y, z, w, v, p, q.\texttt{emppages}(p) \wedge \texttt{emppages}(q)$$
$$\wedge\, \texttt{emprecords}(p, y) \wedge \texttt{emp-num}(y, x_1) \wedge \texttt{emp-name}(y, w)$$
$$\wedge\, \texttt{emprecords}(q, z) \wedge \texttt{emp-num}(z, x_2) \wedge \texttt{emp-name}(z, v)$$
$$\wedge\, \texttt{compare}(w, v).$$

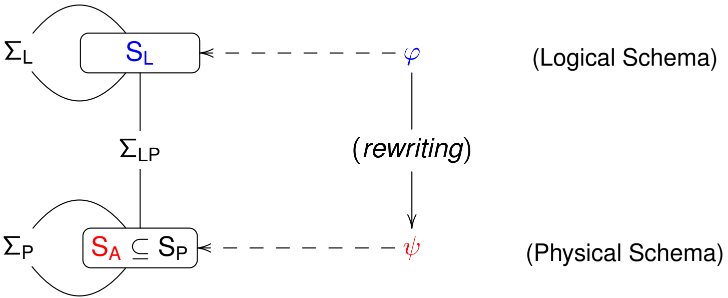
$\Rightarrow$ this plan implements the *block nested loops join* algorithm.

. . . more examples in .

How do We Find PLANS?

Definability and Rewriting

Queries	range-restricted FOL over S_L *definable* w.r.t. Σ and S_A
Ontology/Schema	range-restricted FOL
Data	CWA (complete information for S_A symbols)



Query Plans via Rewriting

Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over $\mathsf{S_A}$:

atomic formula	$\mapsto$	access path (`get-first–get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

$\Rightarrow$ reduces correctness of ψ to logical implication $\Sigma \models \varphi \leftrightarrow \psi$

Non-logical (but necessary) Addons

1. Non-logical properties/operators
   - binding patterns
   - duplication of data and duplicate-preserving/eliminating projections
   - sortedness of data (with respect to the *iterator semantics*) and sorting
2. Cost model

Query Plans via Rewriting

Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over S_A:

atomic formula	$\mapsto$	access path (`get-first–get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

$\Rightarrow$ reduces correctness of ψ to logical implication $\Sigma \models \varphi \leftrightarrow \psi$

Non-logical (but necessary) Add-ons

1. Non-logical properties/operators
   - binding patterns
   - duplication of data and duplicate-preserving/eliminating projections
   - sortedness of data (with respect to the *iterator semantics*) and sorting
2. Cost model

Query Plans via Rewriting

Plans as Formulas

Represent *query plans* as (annotated) range-restricted formulas ψ over $\mathsf{S_A}$:

atomic formula	$\mapsto$	access path (`get-first`–`get-next` iterator)
conjunction	$\mapsto$	nested loops join
existential quantifier	$\mapsto$	projection (annotated w/duplicate info)
disjunction	$\mapsto$	concatenation
negation	$\mapsto$	simple complement

$\Rightarrow$ reduces correctness of ψ to logical implication $\Sigma \models \varphi \leftrightarrow \psi$

Non-logical (but necessary) Add-ons

1. Non-logical properties/operators
   - binding patterns
   - duplication of data and duplicate-preserving/eliminating projections
   - sortedness of data (with respect to the *iterator semantics*) and sorting
2. Cost model

Chase and Backchase

IDEA #1 (Database Theory, CQ/UCQ)

Inference(s): $Q, (\forall \bar{x}.Q_1 \rightarrow Q_2) \vdash Q \cup Q_2\theta$ when $Q_1\theta \subseteq Q$

1. (chase): expand Q "maximally" using constraints
2. (plan) choose a "plan" P from the expansion
3. (backchase): expand P using constraints to contain Q (or fail)

$\rightarrow$ can be extended to UCQ+denial constraints

Views: $V_1(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,y)$
$V_2(x,y) \leftrightarrow \exists u.R(x,u) \wedge R(u,y)$
$V_3(x,y) \leftrightarrow \exists t,u.R(x,t) \wedge R(t,u) \wedge R(u,y)$

Query: $Q(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,v) \wedge R(v,y)$

No SOLUTION (using C&B); rewritings

$\exists z.V_1(x,z) and \forall v.V_2(v,z) \rightarrow V_3(v,y)$ and
$\exists z.V_3(z,y) and \forall v.V_2(v,z) \rightarrow V_1(x,v)$

Chase and Backchase

IDEA #1 (Database Theory, CQ/UCQ)

Inference(s): $Q, (\forall \bar{x}.Q_1 \to Q_2) \vdash Q \cup Q_2\theta$ when $Q_1\theta \subseteq Q$

1. (chase): expand Q “maximally” using constraints
2. (plan) choose a “plan” P from the expansion
3. (backchase): expand P using constraints to contain Q (or fail)

$\Rightarrow$ can be extended to UCQ+denial constraints

Views: $V_1(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,y)$
$V_2(x,y) \leftrightarrow \exists u.R(x,u) \wedge R(u,y)$
$V_3(x,y) \leftrightarrow \exists t,u.R(x,t) \wedge R(t,u) \wedge R(u,y)$

Query: $Q(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,v) \wedge R(v,y)$

No SOLUTION (using C&B); rewritings

$\exists z.V_1(x,z) and \forall v.V_2(v,z) \to V_3(v,y)$ and
$\exists z.V_3(z,y) and \forall v.V_2(v,z) \to V_1(x,v)$

Chase and Backchase

IDEA #1 (Database Theory, CQ/UCQ)

Inference(s): $Q, (\forall \bar{x}.Q_1 \to Q_2) \vdash Q \cup Q_2\theta$ when $Q_1\theta \subseteq Q$

1. (chase): expand Q “maximally” using constraints
2. (plan) choose a “plan” P from the expansion
3. (backchase): expand P using constraints to contain Q (or fail)

$\Rightarrow$ can be extended to UCQ+denial constraints

Views: $V_1(x, y) \leftrightarrow \exists t, u, v. R(t, x) \wedge R(t, u) \wedge R(u, y)$
$V_2(x, y) \leftrightarrow \exists u. R(x, u) \wedge R(u, y)$
$V_3(x, y) \leftrightarrow \exists t, u. R(x, t) \wedge R(t, u) \wedge R(u, y)$

Query: $Q(x, y) \leftrightarrow \exists t, u, v. R(t, x) \wedge R(t, u) \wedge R(u, v) \wedge R(v, y)$

No SOLUTION (using C&B); rewritings

$\exists z. V_1(x, z) and \forall v. V_2(v, z) \to V_3(v, y)$ and
$\exists z. V_3(z, y) and \forall v. V_2(v, z) \to V_1(x, v)$

Chase and Backchase

IDEA #1 (Database Theory, CQ/UCQ)

Inference(s): $Q, (\forall \bar{x}.Q_1 \rightarrow Q_2) \vdash Q \cup Q_2\theta$ when $Q_1\theta \subseteq Q$

1. (chase): expand Q “maximally” using constraints
2. (plan) choose a “plan” P from the expansion
3. (backchase): expand P using constraints to contain Q (or fail)

$\Rightarrow$ can be extended to UCQ+denial constraints

$$
\begin{array}{ll}
\text{Views:} & V_1(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,y) \\
& V_2(x,y) \leftrightarrow \exists u.R(x,u) \wedge R(u,y) \\
& V_3(x,y) \leftrightarrow \exists t,u.R(x,t) \wedge R(t,u) \wedge R(u,y) \\
\text{Query:} & Q(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,v) \wedge R(v,y)
\end{array}
$$

No SOLUTION (using C&B); rewritings

$$
\begin{array}{l}
\exists z.V_1(x,z) and \forall v.V_2(v,z) \rightarrow V_3(v,y) \text{ and} \\
\exists z.V_3(z,y) and \forall v.V_2(v,z) \rightarrow V_1(x,v)
\end{array}
$$

Chase and Backchase

IDEA #1 (Database Theory, CQ/UCQ)

Inference(s): $Q, (\forall \bar{x}.Q_1 \to Q_2) \vdash Q \cup Q_2\theta$ when $Q_1\theta \subseteq Q$

1. (chase): expand Q “maximally” using constraints
2. (plan) choose a “plan” P from the expansion
3. (backchase): expand P using constraints to contain Q (or fail)

$\Rightarrow$ can be extended to UCQ+denial constraints

$$\begin{array}{ll}
\text{Views:} & V_1(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,y) \\
& V_2(x,y) \leftrightarrow \exists u.R(x,u) \wedge R(u,y) \\
& V_3(x,y) \leftrightarrow \exists t,u.R(x,t) \wedge R(t,u) \wedge R(u,y) \\
\text{Query:} & Q(x,y) \leftrightarrow \exists t,u,v.R(t,x) \wedge R(t,u) \wedge R(u,v) \wedge R(v,y)
\end{array}$$

No SOLUTION (using C&B); rewritings

$$\exists z.V_1(x,z) and \forall v.V_2(v,z) \to V_3(v,y) \text{ and}$$
$$\exists z.V_3(z,y) and \forall v.V_2(v,z) \to V_1(x,v)$$

Beth Definability

IDEA #2: What Queries do we allow?

We only allow queries that have *the same answer* in every model of Σ

. . . for a fixed signature S_A (i.e., where the actual data is).

How do we test for this?

φ is *Beth definable* [Beth'56] if

$$\Sigma \cup \Sigma' \models \varphi \rightarrow \varphi'$$

where Σ' (φ') is Σ (φ) in which symbols *NOT in S_A* are *primed*, respectively.

Beth Definability

IDEA #2: What Queries do we allow?

We only allow queries that have *the same answer* in every model of Σ

. . . for a fixed signature S_A (i.e., where the actual data is).

How do we test for this?

φ is *Beth definable* [Beth'56] if

$$\Sigma \cup \Sigma' \models \varphi \rightarrow \varphi'$$

where Σ' (φ') is Σ (φ) in which symbols *NOT in* S_A are *primed*, respectively.

Sequent Calculus: LK

Identity Rules:

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta}\ (\mathit{Axiom}) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta}\ (\mathit{Cut})$$

Logical Rules:

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma, (\neg\varphi) \vdash \Delta}\ (\neg L) \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash (\neg\varphi), \Delta}\ (\neg R)$$

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, (\varphi \vee \psi) \vdash \Delta}\ (\vee L) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash (\varphi \vee \psi), \Delta}\ (\vee R)$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, (\varphi \wedge \psi) \vdash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash (\varphi \wedge \psi), \Delta}\ (\wedge R)$$

Sequent Calculus (with CUT eliminated, for NNF)

Identity Rules:

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta} \; (\mathit{Axiom\ LR})$$

$$\frac{}{\Gamma, \varphi, \neg\varphi \vdash \Delta} \; (\mathit{Axiom\ RR}) \qquad \frac{}{\Gamma \vdash \varphi, \neg\varphi, \Delta} \; (\mathit{Axiom\ LL})$$

Logical Rules:

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, (\varphi \vee \psi) \vdash \Delta} \; (\vee L) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash (\varphi \vee \psi), \Delta} \; (\vee R)$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, (\varphi \wedge \psi) \vdash \Delta} \; (\wedge L) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash (\varphi \wedge \psi), \Delta} \; (\wedge R)$$

Interpolation

How do we find ψ?

If $\Sigma \cup \Sigma' \models \varphi \to \varphi'$ then there is ψ s.t. $\Sigma \cup \Sigma' \models \varphi \to \psi \to \varphi'$ with $\mathcal{L}(\psi) \subseteq \mathcal{L}(\mathsf{S_A})$.

… ψ is called the *Craig Interpolant* [Craig'57].

…we extract an *interpolant* ψ from a (LK) proof of $\Sigma \cup \Sigma' \models \varphi \to \varphi'$

Sequent Calculus (for NNF) and Interpolation

Identity Rules:

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta \leadsto \color{red}{\varphi}}$$

$$\frac{}{\Gamma, \varphi, \neg\varphi \vdash \Delta \leadsto \color{red}{\bot}} \qquad \frac{}{\Gamma \vdash \varphi, \neg\varphi, \Delta \leadsto \color{red}{\top}}$$

Logical Rules:

$$\frac{\Gamma, \varphi \vdash \Delta \leadsto \color{red}{\alpha} \qquad \Gamma, \psi \vdash \Delta \leadsto \color{red}{\beta}}{\Gamma, (\varphi \vee \psi) \vdash \Delta \leadsto \color{red}{\alpha \vee \beta}} \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta \leadsto \color{red}{\alpha}}{\Gamma \vdash (\varphi \vee \psi), \Delta \leadsto \color{red}{\alpha}}$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta \leadsto \color{red}{\alpha}}{\Gamma, (\varphi \wedge \psi) \vdash \Delta \leadsto \color{red}{\alpha}} \qquad \frac{\Gamma \vdash \varphi, \Delta \leadsto \color{red}{\alpha} \qquad \Gamma \vdash \psi, \Delta \leadsto \color{red}{\beta}}{\Gamma \vdash (\varphi \wedge \psi), \Delta \leadsto \color{red}{\alpha \wedge \beta}}$$

LK and Theories

$$
\begin{aligned}
\Sigma \cup \Sigma' \models \varphi \to \varphi' \iff & (\bigwedge \Sigma) \wedge (\bigwedge \Sigma)' \models \varphi \to \varphi' \\
\iff & \models (\bigwedge \Sigma) \to ((\bigwedge \Sigma)' \to (\varphi \to \varphi')) \\
\iff & \models (\bigwedge \Sigma) \to (\varphi \to ((\bigwedge \Sigma)' \to \varphi')) \\
\iff & (\bigwedge \Sigma) \wedge \varphi \models (\bigwedge \Sigma)' \to \varphi') \\
\iff & (\bigwedge \Sigma) \wedge \varphi \models (\bigvee \neg \Sigma)' \vee \varphi' \\
\iff & \Sigma, \varphi \vdash (\neg \Sigma'), \varphi' \quad \text{(due to soundness/completeness)}
\end{aligned}
$$

Not convenient: needs both Σ and negated Σ'!

we use ANALYTIC TABLEAU: a refutation variant of LK to show

$$\Sigma, \Sigma', \varphi, \neg\varphi' \vdash \bot \quad \text{a.k.a. is inconsistent}$$

$\Rightarrow$ need to *tag* left (L)/right(R) formulae to simulate sequents (for interpolation)!

LK and Theories

$$
\begin{aligned}
\Sigma \cup \Sigma' \models \varphi \to \varphi' &\iff (\bigwedge \Sigma) \wedge (\bigwedge \Sigma)' \models \varphi \to \varphi' \\
&\iff \models (\bigwedge \Sigma) \to ((\bigwedge \Sigma)' \to (\varphi \to \varphi')) \\
&\iff \models (\bigwedge \Sigma) \to (\varphi \to ((\bigwedge \Sigma)' \to \varphi')) \\
&\iff (\bigwedge \Sigma) \wedge \varphi \models (\bigwedge \Sigma)' \to \varphi') \\
&\iff (\bigwedge \Sigma) \wedge \varphi \models (\bigvee \neg\Sigma)' \vee \varphi' \\
&\iff \Sigma, \varphi \vdash (\neg\Sigma'), \varphi' \qquad \text{(due to soundness/completeness)}
\end{aligned}
$$

Not convenient: needs both Σ and negated Σ'!

we use ANALYTIC TABLEAU: a refutation variank of LK to show

$$\Sigma, \Sigma', \varphi, \neg\varphi' \vdash \bot \quad \text{a.k.a. is inconsistent}$$

$\Rightarrow$ need to *tag* left (L)/right(R) formulae to simulate sequents (for interpolation)!

LK and Theories

$$
\begin{aligned}
\Sigma \cup \Sigma' \models \varphi \to \varphi' &\iff (\bigwedge \Sigma) \wedge (\bigwedge \Sigma)' \models \varphi \to \varphi' \\
&\iff \models (\bigwedge \Sigma) \to ((\bigwedge \Sigma)' \to (\varphi \to \varphi')) \\
&\iff \models (\bigwedge \Sigma) \to (\varphi \to ((\bigwedge \Sigma)' \to \varphi')) \\
&\iff (\bigwedge \Sigma) \wedge \varphi \models (\bigwedge \Sigma)' \to \varphi') \\
&\iff (\bigwedge \Sigma) \wedge \varphi \models (\bigvee \neg\Sigma)' \vee \varphi' \\
&\iff \Sigma, \varphi \vdash (\neg\Sigma'), \varphi' \qquad \text{(due to soundness/completeness)}
\end{aligned}
$$

Not convenient: needs both Σ and negated Σ'!

we use ANALYTIC TABLEAU: a refutation variank of LK to show

$$\Sigma, \Sigma', \varphi, \neg\varphi' \vdash \bot \quad \text{a.k.a. is inconsistent}$$

$\Rightarrow$ need to *tag* left (L)/right(R) formulae to simulate sequents (for interpolation)!

First-order Variables and Equality

- Quantifier rules
  1. inference rules with Ground constants/terms

Quantifier Rules:

$$\frac{\Gamma, \varphi(t/x) \vdash \Delta}{\Gamma, (\forall x.\varphi) \vdash \Delta}\ (\forall L) \qquad \frac{\Gamma \vdash \varphi(y/x), \Delta}{\Gamma \vdash (\forall x.\varphi), \Delta}\ (\forall R)$$

  2. unification tableau and Skolemization (refutation systems)

- Equality
  1. High-school Axioms (immediate implementation)

$$\vdash x = x$$
$$x = y \wedge \varphi \vdash \varphi(y/x)$$

  2. Superposition rules (efficient implementation)

First-order Variables and Equality

- Quantifier rules
  1. inference rules with Ground constants/terms

Quantifier Rules:

$$\frac{\Gamma, \varphi(t/x) \vdash \Delta}{\Gamma, (\forall x.\varphi) \vdash \Delta}\ (\forall L) \qquad \frac{\Gamma \vdash \varphi(y/x), \Delta}{\Gamma \vdash (\forall x.\varphi), \Delta}\ (\forall R)$$

  2. unification tableau and Skolemization (refutation systems)

- Equality
  1. High-school Axioms (immediate implementation)

$$\vdash x = x$$
$$x = y \wedge \varphi \vdash \varphi(y/x)$$

  2. Superposition rules (efficient implementation)

Issues with TABLEAU

Dealing with the *subformula property* of Tableau

$\Rightarrow$ analytic tableau *explores* formulas *structurally*
$\Rightarrow$ (to large degree) the structure of interpolant
depends on where access paths are present in queries/constraints.

IDEA #3:

Separate *general constraints* from *physical rules* in the formulation of the definability question (and the subsequent interpolant extraction):

$$\Sigma^L \cup \Sigma^R \cup \Sigma^{LR} \models \varphi^L \to \varphi^R \text{ where } \Sigma^{LR} = \{\forall \bar{x}.P^L \leftrightarrow P \leftrightarrow P^R \mid P \in S_A\}$$

Factoring *logical reasoning* from *plan enumeration*

$\Rightarrow$ backtracking tableau to get alternative plans: too slow, too few plans

IDEA #4:

Define *conditional* tableau exploration (using general constraints)
and separate it from plan generation (using physical rules)

Issues with TABLEAU

Dealing with the *subformula property* of Tableau

$\Rightarrow$ analytic tableau *explores* formulas *structurally*

$\Rightarrow$ (to large degree) the structure of interpolant
depends on where access paths are present in queries/constraints.

IDEA #3:

Separate *general constraints* from *physical rules* in the formulation of the definability question (and the subsequent interpolant extraction):

$$\Sigma^L \cup \Sigma^R \cup \Sigma^{LR} \models \varphi^L \rightarrow \varphi^R \text{ where } \Sigma^{LR} = \{\forall \bar{x}.P^L \leftrightarrow P \leftrightarrow P^R \mid P \in \mathsf{S_A}\}$$

Factoring *logical reasoning* from *plan enumeration*

$\Rightarrow$ backtracking tableau to get alternative plans: too slow, too few plans

IDEA #4:

Define *conditional* tableau exploration (using general constraints)
and separate it from plan generation (using physical rules)

Conditional Tableau and Interpolation

Conditional Tableau for $(Q, \Sigma, \mathsf{S_A})$

Proof trees (T^L, T^R): T^L for $\Sigma^L \cup \{Q^L(\bar{a})\}$ over $\{P^L \mid P \in \mathsf{S_A}\}$
T^R for $\Sigma^R \cup \{Q^R(\bar{a}) \to \perp\}$ over $\{P^R \mid P \in \mathsf{S_A}\}$

Closing Set(s)

We call a set C of literals over $\mathsf{S_A}$ a *closing set* for T if, for every branch

1. there is an atom $R(\bar{t})[D]$ such that $D \cup \{\neg R(\bar{t})\} \subseteq C$.
2. there is $\perp[D]$ such that $D \subseteq C$.

$\Rightarrow$ there are many different *minimal* closing sets for T.

Observation

For an arbitrary closing set C, the interpolant for $T^L(T^R)$ is $\perp(\top)$.

Plan Enumeration

Physical Tableau T^P for a Plan P

P	:	L_P	R_P
$R(\bar{t})$	:	$\{\{\neg R^L(\bar{t})\}\}$	$\{\{R^R(\bar{t})\}\}$
$P_1 \wedge P_2$	:	$L_{P_1} \cup L_{P_2}$	$\{S_1 \cup S_2 \mid S_1 \in R_{P_1}, S_2 \in R_{P_2}\}$
$P_1 \vee P_2$	:	$\{S_1 \cup S_2 \mid S_1 \in L_{P_1}, S_2 \in L_{P_2}\}$	$R_{P_1} \cup R_{P_2}$
$\neg P_1$	:	$\{\{L^L(\bar{t}) \mid L^R(\bar{t}) \in S\} \mid S \in R_{P_1}\}$	$\{\{L^R(\bar{t}) \mid L^L(\bar{t}) \in S\} \mid S \in L_{P_1}\}$
$\exists x.P_1$	:	$L_{P_1[t/x]}$	$R_{P_1[t/x]}$

Observation

For a range-restricted formula P over S_A there is an analytic tableau tree T^P that uses only formulæ in $\Sigma^{LR} \cup \{\forall x.\text{true}^R(x)\}$ such that:

1. Open branches of T^P correspond to *sets of literals* $C \in L_P$ (left branch) or $C \in R_P$ (right branch); and
2. The interpolant extracted from the closed tableau $T^P[T^L, T^R]$, the *closure of* (T^L, T^R) by (the *branches* of) T^P, is logically equivalent to P.

Logical&Physical Combined, Controlling the Search

Basic Strategy

1. build (T^L, T^R) for $(Q, \Sigma, \mathsf{S_A})$ to a *certain depth*,
2. build T^P and test if each element in $L_P(R_P)$ closes $T^L(T^R)$.

   if so, $T^P[T^L, T^R]$ is closed tableau yielding an interpolant equivalent to P;
   (...otherwise extend depth in step 1 and repeat.)

NOTE: in step 2 we can "test" many Ps (plan enumeration), but
how do we know which ones to try? while building these bottom-up?

Controlling the Search

- only use the (phys) rule in $T^L(T^R)$ for $R(\bar{t})$ that appears in $T^R(T^L)$,
- only consider *fragments* that help closing (T^L, T^R)
  $\Rightarrow$ this is determined using the minimal closing sets for (T^L, T^R).

... combine with A^* search (among Ps) with respect to a *cost model*.

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists \bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$
\begin{aligned}
Q[\{ R(x_1, \ldots, x_k) \}] &\leftrightarrow Q[R(x_1, \ldots, x_k)] \\
Q[\{ Q_1 \wedge Q_2 \}] &\leftrightarrow Q[\{ Q_1 \} \wedge \{ Q_2 \}] \\
Q[\{ \neg Q_1 \}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg \{ Q_1 \}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{ Q_1 \vee Q_2 \}] &\leftrightarrow Q[\{ Q_1 \} \vee \{ Q_2 \}] \quad \text{if } \Sigma \cup \{ Q[] \} \models Q_1 \wedge Q_2 \rightarrow \bot \\
Q[\{ \exists x . Q_1 \}] &\leftrightarrow Q[\exists x . \{ Q_1 \}] \quad \text{if} \\
& \Sigma \cup \{ Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x] \} \models y_1 \approx y_2
\end{aligned}
$$

... reasoning abstracted: a DL $\mathcal{CFD}_{nc}^{\forall-}$ (a PTIME fragment)

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists \bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$
\begin{aligned}
Q[\{R(x_1,\ldots,x_k)\}] &\leftrightarrow Q[R(x_1,\ldots,x_k)] \\
Q[\{Q_1 \wedge Q_2\}] &\leftrightarrow Q[\{Q_1\} \wedge \{Q_2\}] \\
Q[\{\neg Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg\{Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{Q_1 \vee Q_2\}] &\leftrightarrow Q[\{Q_1\} \vee \{Q_2\}] \quad \text{if } \Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \rightarrow \bot \\
Q[\{\exists x.Q_1\}] &\leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if} \\
& \Sigma \cup \{Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x]\} \models y_1 \approx y_2
\end{aligned}
$$

... reasoning abstracted: a DL $\mathcal{CFD}_{nc}^{\forall-}$ (a PTIME fragment)

Postprocessing: Duplicate Elimination Elimination

IDEA:

Separate the projection operation ($\exists\bar{x}.$) to

- a duplicate preserving projection ($\exists$) and
- an explicit (idempotent) duplicate elimination operator ($\{\cdot\}$).

Use the following rewrites to eliminate/minimize the use of $\{\cdot\}$:

$$
\begin{aligned}
Q[\{R(x_1,\ldots,x_k)\}] &\leftrightarrow Q[R(x_1,\ldots,x_k)] \\
Q[\{Q_1 \wedge Q_2\}] &\leftrightarrow Q[\{Q_1\} \wedge \{Q_2\}] \\
Q[\{\neg Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\neg\{Q_1\}] &\leftrightarrow Q[\neg Q_1] \\
Q[\{Q_1 \vee Q_2\}] &\leftrightarrow Q[\{Q_1\} \vee \{Q_2\}] \quad \text{if } \Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \to \bot \\
Q[\{\exists x.Q_1\}] &\leftrightarrow Q[\exists x.\{Q_1\}] \quad \text{if} \\
&\qquad \Sigma \cup \{Q[] \wedge (Q_1)[y_1/x] \wedge (Q_1)[y_2/x]\} \models y_1 \approx y_2
\end{aligned}
$$

... reasoning abstracted: a DL $\mathcal{CFD}^{\forall-}_{\text{nc}}$ (a PTIME fragment)

Summary

Take Home

While in theory *interpolation* essentially solves the *query rewriting over FO schemas/views* problem, the devil is (as usual) in the details.

[Borgida, de Bruijn, Franconi, Seylan, Straccia, Toman, Weddell: On Finding Query Rewritings under Expressive Constraints. SEBD 2010: 426-437

. . . but an (almost) working system only this year.

1. FO ($\mathcal{DLFDE}$) tableau based interpolation algorithm
   - ⇒ enumeration of plans factored from of tableau reasoning
   - ⇒ extra-logical binding patterns and cost model
2. Post processing (using $\mathcal{CFDI}_{\text{nc}}$ approximation)
   - ⇒ duplicate elimination elimination
   - ⇒ cut insertion
3. Run time
   - ⇒ library of common data/legacy structures+schema constraints
   - ⇒ finger data structures to simulate merge joins et al.

Research Directions and Open Issues

1. Dealing with ordered data? (merge-joins etc.: we have a partial solution)
2. Decidable schema languages (decidable interpolation problem)?
3. More powerful schema languages (inductive types, etc.)?
4. Beyond FO Queries/Views (e.g., count/sum aggregates)?
5. Coding extra-logical bits (e.g., binding patterns, postprocessing, etc.) in the schema itself?
6. Standard Designs (a plan can always be found as in SQL)?
7. Explanation(s) of non-definability?
8. Fine(r)-grained updates?
9. ...

...and, as always, performance, performance, performance!