KR in Database Systems Implementation
(or Life beyond Lite Logics and CQ/UCQ)

David Toman

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

Joint work with Alexander Hudek and Grant Weddell
### The Textbook View

<table>
<thead>
<tr>
<th>Data</th>
<th>represented as an instance of a <em>relational structure</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>access to data via <em>open formulæ</em> (in an appropriate logic)</td>
</tr>
<tr>
<td>Constraints</td>
<td>data integrity enforces by <em>sentences</em> (in the same logic)</td>
</tr>
</tbody>
</table>

⇒ the instance is a *model* of the constraints

### What about `CREATE VIEW` Statements?

View declaration ~ a sentence $\forall x. V(x) \leftrightarrow \varphi$ (in our logic)

where $V$ is a (new) relational symbol and $\varphi$ is a *query*.

### Much Bigger Deal: Physical Data Independence

<table>
<thead>
<tr>
<th>Logical Symbols</th>
<th>user (visible) relations/tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td></td>
</tr>
<tr>
<td>Physical Symbols</td>
<td>data structures (indices)</td>
</tr>
</tbody>
</table>
Data and Constraints: the Database Recap

The Textbook View

Data represented as an instance of a *relational structure*
Queries access to data via *open formulæ* (in an appropriate logic)
Constraints data integrity enforces by *sentences* (in the same logic)

⇒ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration ~ a sentence $\forall x. V(x) \leftrightarrow \varphi$ (in our logic)
where $V$ is a (new) relational symbol and $\varphi$ is a *query*.

Much Bigger Deal: Physical Data Independence

Logical Symbols user (visible) relations/tables
Mapping
Physical Symbols data structures (indices)
Data and Constraints: the Database Recap

The Textbook View

Data represented as an instance of a *relational structure*
Queries access to data via *open formulae* (in an appropriate logic)
Constraints data integrity enforces by *sentences* (in the same logic)

⇒ the instance is a *model* of the constraints

What about \texttt{CREATE VIEW} Statements?

View declaration \sim a sentence $\forall x. V(x) \leftrightarrow \varphi$ (in our logic)
where $V$ is a (new) relational symbol and $\varphi$ is a *query*.

Much Bigger Deal: Physical Data Independence

<table>
<thead>
<tr>
<th>Logical Symbols</th>
<th>user (visible) relations/tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td>data structures (indices)</td>
</tr>
</tbody>
</table>
### The Textbook View

| Data | represented as an instance of a relational structure |
| Queries | access to data via open formulæ (in an appropriate logic) |
| Constraints | data integrity enforces by sentences (in the same logic) |

⇒ the instance is a model of the constraints

### What about `CREATE VIEW` Statements?

View declaration ~ a sentence $\forall x. V(x) \leftrightarrow \varphi$ (in our logic)

where $V$ is a (new) relational symbol and $\varphi$ is a query.

### Much Bigger Deal: Physical Data Independence

| Logical Symbols | user (visible) relations/tables |
| Mapping | C/C++ goo |
| Physical Symbols | data structures (indices) |
Data and Constraints: the Database Recap

The Textbook View

Data represented as an instance of a *relational structure*
Queries access to data via *open formulæ* (in an appropriate logic)
Constraints data integrity enforces by *sentences* (in the same logic)

⇒ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration ∼ a sentence \( \forall x. V(x) \leftrightarrow \varphi \) (in our logic)

where \( V \) is a (new) relational symbol and \( \varphi \) is a *query*.

Much Bigger Deal: Physical Data Independence

Logical Symbols user (visible) relations/tables
Mapping constraints (+ a minimal runtime)
Physical Symbols data structures (indices)
## The KR Way

### Queries and Ontologies

Queries are answered not only w.r.t. *explicit data* \( (A) \) but also w.r.t. *background knowledge* \( (T) \) under OWA

\[ \Rightarrow \text{Ontology-based Data Access (OBDA)} \]

### Example

- Socrates is a MAN  
  (explicit data)
- Every MAN is MORTAL  
  (ontology)

**List all MORTALs** \[ \Rightarrow \{ \text{Socrates} \} \]  
(query)

### How do we answer queries?

Using *logical implication* (to define certain answers):

\[ \text{Ans}(Q, A, T) := \{ Q(a_1, \ldots, a_k) \mid T \cup A \models Q(a_1, \ldots, a_k) \} \]

\[ \Rightarrow \text{answers are ground Q-atoms logically implied by } A \cup T. \]
Queries are answered not only w.r.t. explicit data ($\mathcal{A}$) but also w.r.t. background knowledge ($\mathcal{T}$) under OWA.

$\Rightarrow$ Ontology-based Data Access (OBDA)

Example

- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (ontology)

List all MORTALs $\Rightarrow \{\text{Socrates}\}$ (query)

How do we answer queries?

Using *logical implication* (to define certain answers):

\[
\text{Ans}(Q, \mathcal{A}, \mathcal{T}) := \{Q(a_1, \ldots, a_k) \mid \mathcal{T} \cup \mathcal{A} \models Q(a_1, \ldots, a_k)\}
\]

$\Rightarrow$ answers are *ground Q-atoms* logically implied by $\mathcal{A} \cup \mathcal{T}$.
Good/Standard News

LOGSPACE/PTIME (data complexity) for query answering:
- (U)CQ and
- DL-Lite/$\mathcal{EL}_\bot$/CFD$_{nc}$/$\forall$-

rules-lite (Horn)

Bad News
- no negative queries/sub-queries
- no negations in ABox
- no closed-world assumption
- counter-intuitive query answers
Complexity

Good/Standard News

LOGSPACE/PTIME (data complexity) for query answering:
- (U)CQ and
- DL-Lite/EL/CFD$_{nc}$/“rules”-lite (Horn)

Bad News

- no negative queries/sub-queries
- no negations in ABox
- no closed-world assumption
- counter-intuitive query answers
Difficulties: Unintuitive Answers

Example

- $\text{EMP}(\text{Sue})$
- $\text{EMP} \sqsubseteq \exists \text{PHONENUM}$ (or $\forall x. \text{EMP}(x) \rightarrow \exists y. \text{PHONENUM}(x, y)$)

User: Does Sue have a phone number?
Information System: YES

User: OK, tell me Sue's phone number!
Information System: (no answer)
Difficulties: Unintuitive Answers

Example

- \( EMP(Sue) \)
- \( EMP \sqsubseteq \exists \text{PHONENUM} \quad \text{(or} \quad \forall x. EMP(x) \rightarrow \exists y. \text{PHONENUM}(x, y) \text{)} \)

User: Does Sue have a phone number?

Information System: YES
Difficulties: Unintuitive Answers

Example

- EMP(Sue)
- EMP ⊑∃PHONENUM (or ∀x.EMP(x) → ∃y.PHONENUM(x, y))

User: Does Sue have a phone number?
Information System: YES

User: OK, tell me Sue’s phone number!
Information System: (no answer)
Difficulties: Unintuitive Answers

Example

- EMP(Sue)
- EMP ⊑∃PHONENUM (or ∀x.EMP(x) → ∃y.PHONENUM(x, y))

User: Does Sue have a phone number?

Information System: YES

User: OK, tell me Sue’s phone number!

Information System: (no answer)

User:
## Definability and Rewriting

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>range-restricted FOL (a.k.a. SQL)</td>
</tr>
<tr>
<td>Ontology/Schema</td>
<td>range-restricted FOL $\Sigma := \Sigma^L \cup \Sigma^{LP} \cup \Sigma^P$</td>
</tr>
<tr>
<td>Data</td>
<td>CWA (complete information)</td>
</tr>
</tbody>
</table>
What to do?

**Definability and Rewriting**

<table>
<thead>
<tr>
<th>Queries</th>
<th>range-restricted FOL over $S_L$ definable w.r.t. $\Sigma$ and $S_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology/Schema</td>
<td>range-restricted FOL $\Sigma := \Sigma^L \cup \Sigma^{LP} \cup \Sigma^P$</td>
</tr>
<tr>
<td>Data</td>
<td>CWA (complete information for $S_A$ symbols)</td>
</tr>
</tbody>
</table>

David Toman (et al.) [KR in DBMS Implementation: Definability/Interpolation]
What to do?

Definability and Rewriting

Queries range-restricted FOL over $S_L$ definable w.r.t. $\Sigma$ 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)

- users: looks like a single model (of the logical schema)
- implementation: many models but definable queries answer the same in each of them

Diagram:

```
Query (S_L) → Compiler → Relational Algebra (over S_A) → Evaluator → Answers

Schema (S_L ∪ S_P)

Data (S_A ⊂ S_P)
```
### Definability and Rewriting

<table>
<thead>
<tr>
<th>Queries</th>
<th>range-restricted FOL over $S_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology/Schema</td>
<td>range-restricted FOL $\Sigma := \Sigma^L$</td>
</tr>
<tr>
<td>Data</td>
<td>CWA (complete information for $S_P$)</td>
</tr>
</tbody>
</table>

- **users**: looks like a *single model* (of the logical schema $S_L$)
- **implementation**: many models

**but definable queries answer**

- **Query ($S_L$)**
- **Schema** ($S_L \cup S_P$)
- **Data** ($S_A \subset S_P$)
- **Compiler**
- **Evaluator**
- **Answers**

Relational Algebra (over $S_A$)

---

**Fundamentals of Physical Design and Query Compilation**

David Toman
Grant Weddell

©2011
GRAND UNIFIED APPROACH TO QUERY COMPILATION

PART I: WHAT CAN IT DO?
What can this do?

**GOAL**

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

1. linked data structures, pointers, . . .
2. access to search structures (index access and selection),
3. hash-based access to data (including hash-joins),
4. multi-level storage (aka disk/remote/distributed files), . . .
5. materialized views (FO-definable),
6. updates through logical schema (*needs id invention!*), . . .

. . . all **without** having to code (too much) in C/C++!
What can it do?

GOAL

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

linked data structures, pointers, . . .
access to search structures (index access and selection),
hash-based access to data (including hash-joins),
multi-level storage (aka disk/remote/distributed files), . . .
materialized views (FO-definable),
updates through logical schema (*needs id invention!*), . . .

. . . all *without* having to code (too much) in C/C++!
## Lists and Pointers

### Logical Schema

![Logical Schema Diagram]

### Physical Design

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

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

   - record `dept` of
     - integer `num`
     - string `name`
     - reference `manager`

### Access Paths

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

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

\[
\forall x, y, z. \text{employee}(x, y, z) \rightarrow \exists w. \text{empfile}(w) \land \text{emp-num}(w, x),
\]

\[
\forall a, x. \text{empfile}(a) \land \text{emp-num}(a, x) \rightarrow \exists y, z. \text{employee}(x, y, z), ...
\]
What can this do: navigating pointers

Example queries:

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

   \[\exists a.\text{empfile}(a) \land \text{emp-num}(a, x) \land \text{emp-name}(a, y)\]
What can this do: navigating pointers

Example queries:

1. List all employee numbers and names ($\exists z, w.\text{employee}(x, y, z, w)$):
   $\exists a.\text{empfile}(a) \land \text{emp-num}(a, x) \land \text{emp-name}(a, y)$

2. List all department numbers with their manager names
   ($\exists z, u, v, w.\text{department}(x, z, u) \land \text{employee}(u, y, v, w)$):
   $\exists a.\text{empfile}(a) \land \text{emp-name}(a, y) \land \text{emp-dept}(a, d) \land \text{dept-num}(d, x) \land \text{dept-mgr}(d, e) \land \text{emp-name}(e, y) \land \text{compare}(a, b)$
What can this do: navigating pointers

Example queries:

1. List all employee numbers and names (\(\exists z, w.\text{employee}(x, y, z, w)\)):
   \[
   \exists a.\text{empfile}(a) \land \text{emp-num}(a, x) \land \text{emp-name}(a, y)
   \]

2. List all department numbers with their manager names
   (\(\exists z, u, v, w.\text{department}(x, z, u) \land \text{employee}(u, y, v, w)\)):
   \[
   \exists a, d, e.\text{empfile}(a) \land \text{emp-dept}(a, d) \\
   \land \text{dept-num}(d, x) \land \text{dept-mgr}(d, e) \land \text{emp-name}(e, y) \\
   \Rightarrow \text{needs “departments have at least one employee”}.
   \]
What can this do: navigating pointers

Example queries:

1. List all employee numbers and names \((\exists z, w.\text{employee}(x, y, z, w))\):
   \[
   \exists a.\text{empfile}(a) \land \text{emp-num}(a, x) \land \text{emp-name}(a, y)
   \]

2. List all department numbers with their manager names
   \((\exists z, u, v, w.\text{department}(x, z, u) \land \text{employee}(u, y, v, w))\):
   \[
   \exists a, d, e.\text{empfile}(a) \land \text{emp-dept}(a, d) \\
   \land \text{dept-num}(d, x) \land \text{dept-mgr}(d, e) \land \text{emp-name}(e, y)
   \]
   \[
   \Rightarrow \text{needs “departments have at least one employee”}.
   \]
   \[
   \exists a, b, d.\text{empfile}(a) \land \text{emp-name}(a, y) \land \text{emp-dept}(a, d) \\
   \land \text{dept-num}(d, x) \land \text{dept-mgr}(d, b) \land \text{compare}(a, b)
   \]
   \[
   \Rightarrow \text{needs “managers work in their own departments”}.
   \]
What can this do: navigating pointers

Example queries:

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

\[ \exists a. \text{empfile}(a) \land \text{emp-num}(a, x) \land \text{emp-name}(a, y) \]

2 List all department numbers with their manager names

\((\exists z, u, v, w.\text{department}(x, z, u) \land \text{employee}(u, y, v, w))\):

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

\[ \Rightarrow \text{needs “departments have at least one employee”}. \]

\[ \ldots \text{needs duplicate elimination during projection}. \]

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

\[ \Rightarrow \text{needs “managers work in their own departments”}. \]

\[ \ldots \text{NO duplicate elimination during projection}. \]
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.

List all employees with the same name

\((\exists z, u, v, w, t. \text{employee}(x_1, z, u, v) \land \text{employee}(x_2, z, w, t)):\)

\[ \exists y, z, w, v, p, q. \text{emppages}(p) \land \text{emppages}(q) \]

\[ \quad \land \text{emprecords}(p, y) \land \text{emp-num}(y, x_1) \land \text{emp-name}(y, w) \]

\[ \quad \land \text{emprecords}(q, z) \land \text{emp-num}(z, x_2) \land \text{emp-name}(z, v) \]

\[ \quad \land \text{compare}(w, v). \]

⇒ this plan implements the `block nested loops join` algorithm.

...many more examples in ...
GRAND UNIFIED APPROACH TO QUERY COMPILATION

PART II: HOW DOES IT WORK?
Definability and Rewriting

Queries
- range-restricted FOL over $S_L$ *definable* w.r.t. $\Sigma$ and $S_A$

Ontology/Schema
- range-restricted FOL

Data
- CWA (complete information for $S_A$ symbols)

---

*$\Sigma_L$ (Logical Schema)

*$\Sigma_{LP}$

*$\Sigma_P$

*$S_L$ (Logical Schema)

*$S_A \subseteq S_P$ (Physical Schema)

*$\varphi$ (rewriting)

*$\psi$ (Physical Schema)
IDEA #1:

Represent physical design as access paths \((S_A)\) and constraints \((\Sigma)\). Represent query plans as (annotated) range-restricted formulas \(\psi\) over \(S_A\).

- atomic formula \(\mapsto\) access path
- conjunction \(\mapsto\) nested loops join
- existential quantifier \(\mapsto\) projection (annotated w/ duplicate info)
- disjunction \(\mapsto\) concatenation
- negation \(\mapsto\) simple complement
**Query Plans via Interpolation**

**IDEA #1:**

Represent *physical design* as *access paths* ($S_A$) and constraints ($\Sigma$). Represent *query plans* as (annotated) range-restricted formulas $\psi$ over $S_A$.

$\Rightarrow$ reduces correctness of $\psi$ w.r.t. the user query $\varphi$ to $\Sigma \models \varphi \leftrightarrow \psi$
IDEA #1:
Represent *physical design* as *access paths* $(S_A)$ and constraints $(\Sigma)$. Represent *query plans* as (annotated) range-restricted formulas $\psi$ over $S_A$.

⇒ reduces correctness of $\psi$ w.r.t. the user query $\varphi$ to $\Sigma \models \varphi \iff \psi$

IDEA #2:
Use *interpolation* to search for $\psi$:
extract an *interpolant* $\psi$ from a (TABLEAU) proof of $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$
Query Plans via Interpolation

**IDEA #1:**

Represent *physical design* as *access paths* ($S_A$) and *constraints* ($\Sigma$). Represent *query plans* as (annotated) range-restricted formulas $\psi$ over $S_A$.

$\Rightarrow$ reduces correctness of $\psi$ w.r.t. the user query $\varphi$ to $\Sigma \models \varphi \iff \psi$

**IDEA #2:**

Use *interpolation* to search for $\psi$:

extract an *interpolant* $\psi$ from a (TABLEAU) proof of $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

$\Rightarrow$ *Beth Definability* of $\varphi$ over $\Sigma$ and $S_A$ resolves the existence of $\psi$

(except for *binding patterns*)
Subformula (structural) Property: not enough rewritings (plans)

- $\Sigma^L \cup \Sigma^R \cup \Sigma^{LR} \models \varphi^L \rightarrow \varphi^R$ where $\Sigma^{LR} = \{ \forall \bar{x}.P^L \leftrightarrow P \leftrightarrow P^R | P \in S_A \}$
Subformula (structural) Property: not enough rewritings (plans)

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

Alternative Proofs/Plans: backtracking is too slow

- *conditional formulae*: $\varphi[C]$ where $C$ is a set of (ground) literals over $S_A$
- logical (non-backtrackable) *conditional tableau* ($T^L, T^R$)
- cost-based plan enumeration based on *closing sets* in ($T^L, T^R$) and $\Sigma^{LR}$
Engineering Issues

Subformula (structural) Property: not enough rewritings (plans)

- \( \Sigma^L \cup \Sigma^R \cup \Sigma^{LR} \models \varphi^L \rightarrow \varphi^R \) where \( \Sigma^{LR} = \{ \forall \bar{x}.P^L \leftrightarrow P \leftrightarrow P^R \mid P \in S_A \} \)

Alternative Proofs/Plans: backtracking is too slow

- *conditional formulæ*: \( \varphi[C] \) where \( C \) is a set of (ground) literals over \( S_A \)
- logical (non-backtrackable) *conditional tableau* \( (T^L, T^R) \)
- cost-based plan enumeration based on *closing sets* in \( (T^L, T^R) \) and \( \Sigma^{LR} \)

Non-logical Features: dealing with duplicates et al.

- \( Q[\exists x. Q_1] \leftrightarrow Q[\exists x. Q_1] \) if \( \Sigma \cup \{Q\} \land Q_1[y_1/x] \land Q_1[y_2/x] \) \( \models y_1 \approx y_2 \)
- \( Q[Q_1 \lor Q_2] \leftrightarrow Q[Q_1 \lor Q_2] \) if \( \Sigma \cup \{Q\} \models Q_1 \land Q_2 \rightarrow \bot \)
  \[ \Rightarrow CFDL_{nc} \text{ description logic approximation of } \Sigma \text{ (PTIME reasoning)}. \]

... for details see ...
Summary of the Approach

1. FO ($\mathcal{DLFDE}$) tableau based interpolation algorithm
   ⇒ enumeration of plans factored from reasoning
   ⇒ range-restricted queries and constraints → ground terms only
   ⇒ extra-logical binding patterns and cost model

2. Post processing (using $\mathcal{CFDL}_{nc}$ approximation)
   ⇒ duplicate elimination elimination
   ⇒ cut insertion

3. Run time
   ⇒ library of common data structures+schema constraints
      or an interface to a legacy system
   ⇒ 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!
Message from our Sponsors

Database Group at the University of Waterloo

- 7 professors, affiliated faculty, postdocs, 30+ graduate students, ...
- wide range of research interests
  - Advanced query processing/Knowledge representation
  - System aspects of database systems and Distributed data management
  - Data quality/Managing uncertain data/Data mining
  - New(-ish) domains (text, streaming, graph data/RDF, OLAP)
- research sponsored by governments, and local/global companies
  - NSERC/CFI/OIT and Google, IBM, SAP, OpenText, ...
- part of a School of CS with 75+ professors, 300+ grad students, etc.
  - AI&ML, Algorithms&Data Structures, PL, Theory, Systems, ...

Cheriton School of Computer Science has been ranked #18 in CS by the world by US News and World Report (#1 in Canada).

... and we are always looking for good graduate students (MMath/PhD)

⇒ comes with full support over multiple years