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

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)

$\Rightarrow$ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration $\sim$ a sentence $\forall \mathbf{x}. V(\mathbf{x}) \leftrightarrow \varphi$ (in our logic)
where V is a (new) relational symbol and φ 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 formulæ* (in an appropriate logic)
Constraints	data integrity enforces by *sentences* (in the same logic)

$\Rightarrow$ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration $\sim$ a sentence $\forall \mathbf{x}.V(\mathbf{x}) \leftrightarrow \varphi$ (in our logic)
where V is a (new) relational symbol and φ 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 formulæ* (in an appropriate logic)
Constraints	data integrity enforces by *sentences* (in the same logic)

$\Rightarrow$ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration $\sim$ a sentence $\forall \mathbf{x}. V(\mathbf{x}) \leftrightarrow \varphi$ (in our logic)
where V is a (new) relational symbol and φ 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 formulæ* (in an appropriate logic)
Constraints	data integrity enforces by *sentences* (in the same logic)

$\Rightarrow$ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

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

$\Rightarrow$ the instance is a *model* of the constraints

What about `CREATE VIEW` Statements?

View declaration $\sim$ a sentence $\forall \mathbf{x}. V(\mathbf{x}) \leftrightarrow \varphi$ (in our logic)

where V is a (new) relational symbol and φ 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* ($\mathcal{A}$)
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 $\Rightarrow$ $\{$Socrtes$\}$ (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 $\mathcal{A} \cup \mathcal{T}$.

The KR Way

Queries and Ontologies

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 $\Rightarrow$ {Socrtes} (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 $\mathcal{A} \cup \mathcal{T}$.

Complexity

Good/Standard News

LOGSPACE/PTIME (data complexity) for query answering:

- (U)CQ and
- DL-Lite/$\mathcal{EL}_\perp$/$\mathcal{CFD}^\forall_{\rm nc}$/"rules"-lite (Horn)

Bad News

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

University of Waterloo

Complexity

Good/Standard News

LOGSPACE/PTIME (data complexity) for query answering:

- (U)CQ and
- DL-Lite/$\mathcal{EL}_{\perp}$/$\mathcal{CFD}^{\forall}_{\mathrm{nc}}$/“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

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

Difficulties: Unintuitive Answers

Example

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

User: *Does Sue have a phone number?*

Information System: *YES*

Difficulties: Unintuitive Answers

Example

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

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

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

User: *Does Sue have a phone number?*

Information System: *YES*

User: *OK, tell me Sue's phone number!*

Information System: *(no answer)*

User:

What to do?

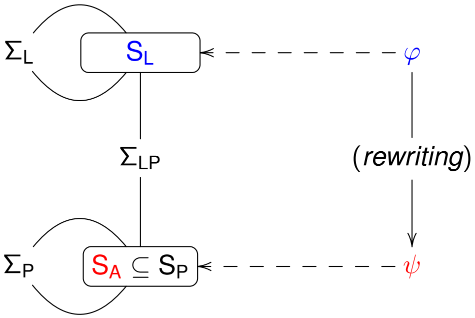
Definability and Rewriting

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)

What to do?

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)



Logical Schema
and User Queries

Physical Schema
and Query Plans

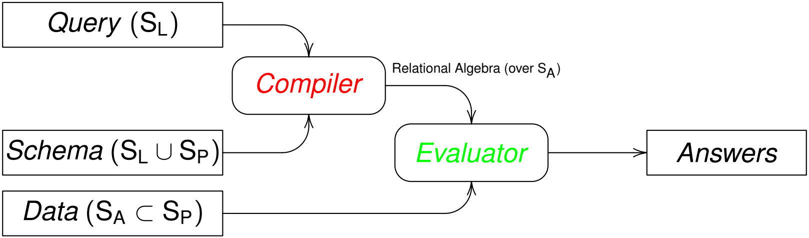
What to do?

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)

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

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



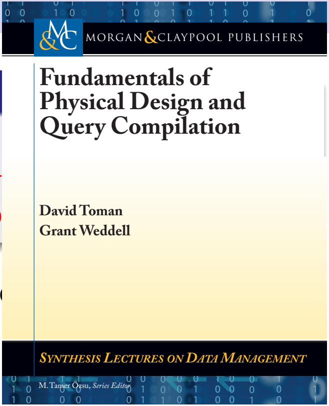
What to do?

Definability and Rewriting

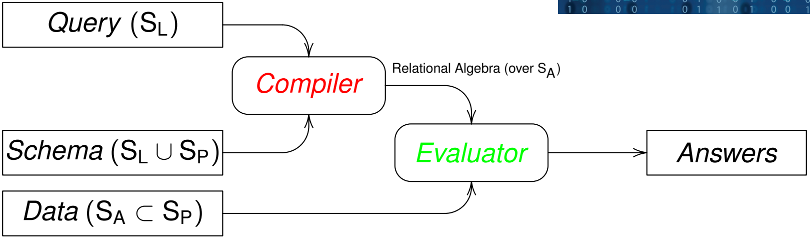
Queries	range-restricted FOL over S_L
Ontology/Schema	range-restricted FOL $\Sigma := \Sigma^L$
Data	CWA (complete information fo

- users: looks like a *single model* (of the logical s
- implementation: many models

but *definable* queries answer



©2011



University of Waterloo

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 this do?

GOAL

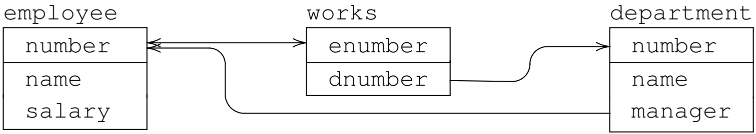
Generate query plans *that compete with hand-written programs in C*

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++ !

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.,

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

What can this do: navigating pointers

Example queries:

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

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

What can this do: navigating pointers

Example queries:

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

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

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

What can this do: navigating pointers

Example queries:

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

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

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

What can this do: navigating pointers

Example queries:

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 ($\exists z, u, v, w.\texttt{department}(x, z, u) \wedge \texttt{employee}(u, y, v, w)$):

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

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

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

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

What can this do: navigating pointers

Example queries:

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

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

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

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

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

... needs *duplicate elimination* during projection.

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

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

... NO *duplicate elimination* during projection.

What can this do: two-level store

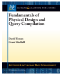
The access path `empfile` is refined by `emppages`/1/0 and `emprecords`/2/1:

- `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
$(\exists z, u, v, w, t.\mathtt{employee}(x_1, z, u, v) \wedge \mathtt{employee}(x_2, z, w, t))$:

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

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

. . . many more examples in 

.

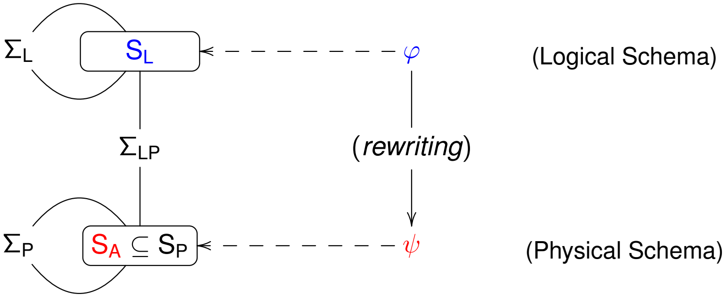
GRAND UNIFIED APPROACH TO QUERY COMPILATION

PART II: HOW DOES IT WORK?

The Plan

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 Interpolation

IDEA #1:

Represent *physical design* as *access paths* (S_A) and constraints (Σ).
Represent *query plans* as (annotated) range-restricted formulas ψ 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 (Σ).
Represent *query plans* as (annotated) range-restricted formulas ψ over S_A.

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

Query Plans via Interpolation

IDEA #1:

Represent *physical design* as *access paths* (S_A) and constraints (Σ).
Represent *query plans* as (annotated) range-restricted formulas ψ over S_A.

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

IDEA #2:

Use *interpolation* to search for ψ:
extract an *interpolant* ψ 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 (Σ).
Represent *query plans* as (annotated) range-restricted formulas ψ over S_A.

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

IDEA #2:

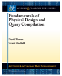
Use *interpolation* to search for ψ:
extract an *interpolant* ψ from a (TABLEAU) proof of $\Sigma \cup \Sigma^* \models \varphi \rightarrow \varphi^*$

$\Rightarrow$ *Beth Definability* of φ over Σ and S_A resolves the existence of ψ
(except for *binding patterns*)

Engineering Issues

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

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

. . . for details see .

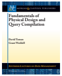
Engineering Issues

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

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

Alternative Proofs/Plans: backtracking is too slow

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

. . . for details see 

.

University of Waterloo

Engineering Issues

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

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

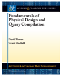
Alternative Proofs/Plans: backtracking is too slow

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

Non-logical Features: dealing with duplicates et al.

- $Q[\exists x.Q_1] \mapsto Q[\exists x.Q_1]$ if $\Sigma \cup \{Q[] \wedge Q_1[y_1/x] \wedge Q_1[y_2/x]\} \models y_1 \approx y_2$
- $Q[Q_1 \vee Q_2] \mapsto Q[Q_1 \vee Q_2]$ if $\Sigma \cup \{Q[]\} \models Q_1 \wedge Q_2 \to \bot$

$\Rightarrow \mathcal{CFDI}_{\mathsf{nc}}$ description logic approximation of Σ (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 $\mathcal{CFDI}_{\text{nc}}$ 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)

$\Rightarrow$ comes with full support over multiple years

University of Waterloo