Logical Approach to Physical Data Independence and Query Compilation

Advanced Physical Designs

David Toman

D.R. Cheriton School of Computer Science
University of Waterloo
The Story So Far…

1. Physical Data Independence (OBDA, Data Exchange, …)
2. Logic-based formalization (Relational model, constraints)
3. Queries and Answers

\[
\text{cert}_{\Sigma,D}(\varphi) = \{ \bar{a} \mid \Sigma \cup D \models \varphi(\bar{a})\} = \bigcap_{l \models \Sigma \cup D} \{ \bar{a} \mid l \models \varphi(\bar{a})\}
\]

4. Only queries \textit{logically equivalent} to range-restricted queries over \(S_A\).

- what does this kind of arrangement allow?
- why is this \textit{efficient}?
- how to find such equivalent queries
ADVANCED PHYSICAL DESIGNS
Case Studies

- Main-memory pointers
- Hash tables, linked lists, et al.
- Built-in operations
- Two-level store
Main Memory and Pointers

Logical Schema:

```
employee
  number
  name
  salary

works
  enumber
  dnumber

department
  number
  name
  manager
```

Advanced Physical Designs 5 / 14
Main Memory and Pointers

Logical Schema:

employee
number
name
salary

works

employee
number
name
dnumber

department
number
name
manager

Physical Schema:

record emp of
integer num
string name
integer salary
reference dept

record dept of
integer num
string name
reference manager

... and an array holding emp records (called empfile).
Logical Schema & Constraints:

\[ S_L = \{ \text{employee/3, department/3, works/2} \}; \]
\[ \Sigma_L = \{ \forall x_1, x_2, y_1, y_2. \exists z. (\text{employee}(z, x_1, x_2) \land \text{employee}(z, y_1, y_2)) \rightarrow ((x_1 = y_1) \land (x_2 = y_2)), \]
\[ \forall x, y, z. (\text{works}(z, x) \land \text{works}(z, y)) \rightarrow (x = y), \]
\[ \forall x, y, z. \text{department}(y, z, x) \rightarrow \exists u, v. \text{employee}(x, u, v), \ldots \} \]
Main Memory and Pointers: Formalization

**Logical Schema&Constraints:**

\[ S_L = \{ \text{employee/3, department/3, works/2} \}; \]

\[ \Sigma_L = \{ \forall x_1, x_2, y_1, y_2. \exists z. (\text{employee}(z, x_1, x_2) \land \text{employee}(z, y_1, y_2)) \rightarrow ((x_1 = y_1) \land (x_2 = y_2)), \]

\[ \forall x, y, z. (\text{works}(z, x) \land \text{works}(z, y)) \rightarrow (x = y), \]

\[ \forall x, y, z. \text{department}(y, z, x) \rightarrow \exists u, v. \text{employee}(x, u, v), \ldots \} \]

**Physical Schema&Constraints:**

\[ S_A = \{ \text{empfile/1/0, emp-num/2/1, }\]

\[ \text{emp-name/2/1, emp-salary/2/1, emp-dept/2/1, }\]

\[ \text{dept-num/2/1, dept-name/2/1, dept-manager/2/1}, \]

\[ \Sigma_{LP} = \{ \forall x. (\text{empfile}(x) \rightarrow \exists y. \text{emp-num}(x, y)), \ldots, \]

\[ \forall x, y. (\text{emp-dept}(x, y) \rightarrow \text{deptfile}(y)), \]

\[ \forall x. (\text{deptfile}(x) \rightarrow \exists y. \text{dept-num}(x, y)), \ldots, \]

\[ \forall x, y. (\text{dept-manager}(x, y) \rightarrow \text{empfile}(y)), \]

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

\[ \forall x, y, z, w. ((\text{empfile}(w) \land \text{emp-num}(w, x) \land \text{emp-name}(w, y) \land \text{emp-salary}(w, z)) \rightarrow \text{employee}(x, y, z), \ldots \} \]
∃z. employee(x, y, z):
E?z.Employee(x, y, ?z)
Plan: 26 (15n, 5n)
E?x1.(Empfile(?x1)^Emp-num(?x1,x)^Emp-name(?x1,y))
Main Memory and Pointers: Queries and Plans

1. $\exists z. \text{employee}(x, y, z)$:
   
   E?z. Employee(x, y, ?z)
   
   Plan: 26 (15n, 5n)
   
   E?x1. (Empfile(?x1)^Emp-num(?x1, x)^Emp-name(?x1, y))

2. Department(x, y, z):

   Department(x, y, z)
   
   Plan: 241 (35n, 5n)
   
   E?x2. (Empfile(?x2)^Emp-num(?x2, z)^E?x1. (Emp-dept(?x2, ?x1)
       ^Dept-name(?x1, y)^Dept-num(?x1, x)
       ^E?s0. (Dept-manager(?x1, ?s0)^Cmp(?x2, ?s0))))

3.∃ y, v, w. employee(x1, x2, y) ∧ works(x1, v) ∧ department(v, x3, w):

   E?y, ?v, ?w. Employee(x1, x2, ?y)^Works(x1, ?v)^Department(?v, x3, ?w)
   
   Plan: 50 (40n, 5n)
   
   E?x5. (Empfile(?x5)^Emp-num(?x5, x1)^Emp-name(?x5, x2)
       ^E?x4. (Emp-dept(?x5, ?x4)^Dept-name(?x4, x3)))
∃z. \texttt{employee}(x, y, z):
E?z. \texttt{Employee}(x, y, ?z)
Plan: 26 (15n, 5n)
E?x1. (\texttt{Empfile}(?x1) \wedge \texttt{Emp-num}(?x1, x) \wedge \texttt{Emp-name}(?x1, y))

\textbf{Department}(x, y, z):
Department(x, y, z)
Plan: 241 (35n, 5n)
E?x2. (\texttt{Empfile}(?x2) \wedge \texttt{Emp-num}(?x2, z) \wedge E?x1. (\texttt{Emp-dept}(?x2, ?x1) \\
\wedge \texttt{Dept-name}(?x1, y) \wedge \texttt{Dept-num}(?x1, x) \\
\wedge E?s0. (\texttt{Dept-manager}(?x1, ?s0) \wedge \texttt{Cmp}(?x2, ?s0))))

Is there a \textit{shorter} plan?
∃z. employee(x, y, z):
E?z. Employee(x, y, ?z)
Plan: 26 (15n, 5n)
E?x1. (Empfile(?x1)^Emp-num(?x1, x)^Emp-name(?x1, y))

Department(x, y, z):
Department(x, y, z)
Plan: 241 (35n, 5n)
E?x2. (Empfile(?x2)^Emp-num(?x2, z)^E?x1. (Emp-dept(?x2, ?x1)
^Dept-name(?x1, y)^Dept-num(?x1, x)
^E?s0. (Dept-manager(?x1, ?s0)^Cmp(?x2, ?s0))))

Is there a shorter plan? YES:
E?x2. (Empfile(?x2)^E?x1. (Emp-dept(?x2, ?x1)
^Dept-name(?x1, y)^Dept-num(?x1, x)
^E?x3. (Dept-manager(?x1, ?x3)^Emp-num(?x3, z)))
∃ z. employee(x, y, z):
   E?z.Employee(x, y, ?z)
Plan: 26 (15n, 5n)
E?x1.(Empfile(?x1)^Emp-num(?x1, x)^Emp-name(?x1, y))

Department(x, y, z):
   Department(x, y, z)
Plan: 241 (35n, 5n)
E?x2.(Empfile(?x2)^Emp-num(?x2, z)^E?x1.(Emp-dept(?x2, ?x1)
   ^Dept-name(?x1, y)^Dept-num(?x1, x)
   ^E?s0.(Dept-manager(?x1, ?s0)^Cmp(?x2, ?s0))))

Is there a shorter plan? YES:

   E?x2.(Empfile(?x2)^E?x1.(Emp-dept(?x2, ?x1)
   ^Dept-name(?x1, y)^Dept-num(?x1, x)
   ^E?x3.(Dept-manager(?x1, ?x3)^Emp-num(?x3, z)))

⇒ is it better?
∃z.\textit{employee}(x, y, z):
\begin{align*}
\text{E}\ ?z.\text{Employee}(x, y, \ ?z) \\
\text{Plan: 26 (15n, 5n)} \\
\text{E}\ ?x_1. (\text{Empfile}(\ ?x_1) \land \text{Emp-num}(\ ?x_1, x) \land \text{Emp-name}(\ ?x_1, y))
\end{align*}

Department(x, y, z):
\begin{align*}
\text{Department}(x, y, z) \\
\text{Plan: 241 (35n, 5n)} \\
\text{E}\ ?x_2. (\text{Empfile}(\ ?x_2) \land \text{Emp-num}(\ ?x_2, z) \land \text{E}\ ?x_1. (\text{Emp-dept}(\ ?x_2, \ ?x_1) \\
\quad \land \text{Dept-name}(\ ?x_1, y) \land \text{Dept-num}(\ ?x_1, x) \\
\quad \land \text{E}\ ?s_0. (\text{Dept-manager}(\ ?x_1, \ ?s_0) \land \text{Cmp}(\ ?x_2, \ ?s_0)))
\end{align*}

Is there a \textit{shorter} plan? \textbf{YES}:
\begin{align*}
\text{E}\ ?x_2. (\text{Empfile}(\ ?x_2) \land \text{E}\ ?x_1. (\text{Emp-dept}(\ ?x_2, \ ?x_1) \\
\quad \land \text{Dept-name}(\ ?x_1, y) \land \text{Dept-num}(\ ?x_1, x) \\
\quad \land \text{E}\ ?x_3. (\text{Dept-manager}(\ ?x_1, \ ?x_3) \land \text{Emp-num}(\ ?x_3, z))
\end{align*}

\Rightarrow \text{is it better? NO (duplicate elimination)}
Main Memory and Pointers: Queries and Plans

1. $\exists z. \text{employee}(x, y, z)$:
   $E?z. \text{Employee}(x, y, ?z)$
   Plan: 26 (15n, 5n)
   $E?x1. (\text{Empfile}(?x1)^\text{Emp-num}(?x1, x)^\text{Emp-name}(?x1, y))$

2. Department$(x, y, z)$:
   Department$(x, y, z)$
   Plan: 241 (35n, 5n)
   $E?x2. (\text{Empfile}(?x2)^\text{Emp-num}(?x2, z)^E?x1. (\text{Emp-dept}(?x2, ?x1)^\text{Dept-name}(?x1, y)^\text{Dept-num}(?x1, x)^E?s0. (\text{Dept-manager}(?x1, ?s0)^\text{Cmp}(?x2, ?s0))))$

3. $\exists y, v, w. \text{employee}(x1, x2, y) \land \text{works}(x1, v) \land \text{department}(v, x3, w)$:
   $E?y, ?v, ?w. \text{Employee}(x1, x2, ?y)^\text{Works}(x1, ?v)^\text{Department}(?v, x3, ?w)$
   Plan: 50 (40n, 5n)
   $E?x5. (\text{Empfile}(?x5)^\text{Emp-num}(?x5, x1)^\text{Emp-name}(?x5, x2)^E?x4. (\text{Emp-dept}(?x5, ?x4)^\text{Dept-name}(?x4, x3)))$
Hashing, Lists, et al.

Hashing with (list-based) Separate Chaining

Hash Array | Separate Chaining Linked Lists | DeptFile

\[ i : \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bot \rightarrow D_1 \]
\[ j : \bot \rightarrow D_3 \]
\[ n : \bullet \rightarrow \bullet \rightarrow \bot \rightarrow D_2 \]
Hashing, Lists, et al.

Access paths:
\[ S_A \supseteq \{ \text{hash/2/1, hasharraylookup/2/1, listscan/2/1} \} \]

Physical Constraints:
\[ \Sigma_{LP} \supseteq \{ \forall x, y. (\text{deptfile}(x) \land \text{dept-name}(x, y)) \rightarrow \exists z, w. (\text{hash}(y, z) \land \text{hasharraylookup}(z, w) \land \text{listscan}(w, x)), \]
\[ \forall x, y. (\text{hash}(x, y) \rightarrow \exists z. \text{hasharraylookup}(y, z)), \]
\[ \forall x, y. (\text{listscan}(x, y) \rightarrow \text{deptfile}(y)) \]
Access paths:

⇒ \( S_A \supseteq \{\text{hash/2/1}, \text{hasharraylookup/2/1}, \text{listscan/2/1}\} \).

Physical Constraints:

⇒ \( \Sigma_{LP} \supseteq \{\forall x, y.((\text{deptfile}(x) \land \text{dept-name}(x, y)) \rightarrow \exists z, w.(\text{hash}(y, z) \land \text{hasharraylookup}(z, w) \land \text{listscan}(w, x))), \forall x, y.(\text{hash}(x, y) \rightarrow \exists z.\text{hasharraylookup}(y, z)), \forall x, y.(\text{listscan}(x, y) \rightarrow \text{deptfile}(y)) \} \)

Queries:

⇒ \( \exists y, z.((\text{department}(x_1, p, y) \land \text{employee}(y, x_2, z))\{p\}) \).
Built-in Operations

How do we introduce *built-in* functions/operations such as *comparisons, arithmetic, string manipulation*, etc.?

**IDEA**

Make built-in functions into access paths with appropriate binding pattern.

**Example (Integer Inequalities)**

Logical Schema:

\[
\begin{align*}
\text{less} &\leq \text{less}\in S \\
\Rightarrow &\subseteq \Sigma \\
\end{align*}
\]

Physical Schema:

\[
\begin{align*}
\text{less}\in S &\Rightarrow \forall x, y. (x < y) \leftrightarrow \text{less}(x, y) \\
\forall x, y. (x \leq y) &\leftrightarrow \neg \text{less}(y, x) \\
\end{align*}
\]

**Code:**

function less-first function less-next
return (x1 < x2) return false

⇒ we already have cmp/2/2 for equality!
Built-in Operations

How do we introduce *built-in* functions/operations such as *comparisons*, *arithmetic*, *string manipulation*, etc.?

**IDEA**

Make *built in* functions into *access paths* with appropriate binding pattern.
How do we introduce built-in functions/operations such as comparisons, arithmetic, string manipulation, etc.?

**IDEA**

Make built in functions into access paths with appropriate binding pattern.

**Example (Integer Inequalities)**

Logical Schema: $\langle /2, \leq /2 \subseteq S_L$ (written conventionally in infix)

Code:

```plaintext
function less-first function less-next
return (x1 < x2) return false
```

We already have cmp $/2$ $/2$ for equality!
How do we introduce built-in functions/operations such as comparisons, arithmetic, string manipulation, etc.?

**IDEA**

Make built in functions into access paths with appropriate binding pattern.

**Example (Integer Inequalities)**

**Logical Schema:** \( < \{2, \leq \{2 \subseteq S_L \) (written conventionally in infix)

**Physical Schema:** \( \text{less} \{2/2 \in S_A \)

\( \Rightarrow \Sigma_{LP} \supseteq \{ \forall x, y. (x < y) \leftrightarrow \text{less}(x, y) \)

\( \forall x, y. (x \leq y) \leftrightarrow \neg \text{less}(y, x) \} \)
How do we introduce built-in functions/operations such as comparisons, arithmetic, string manipulation, etc.?

**IDEA**

Make built in functions into access paths with appropriate binding pattern.

**Example (Integer Inequalities)**

**Logical Schema:** $< /2, \leq /2 \subseteq S_L$ (written conventionally in infix)

**Physical Schema:** $\text{less}/2/2 \in S_A$

$\Rightarrow \Sigma_{LP} \supseteq \{ \forall x, y. (x < y) \leftrightarrow \text{less}(x, y) \}
\forall x, y. (x \leq y) \leftrightarrow \neg \text{less}(y, x) \}$

**Code:**

```
function less-first
    return (x1 < x2)
```

```
function less-next
    return false
```

we already have cmp /2 /2 for equality!
How do we introduce built-in functions/operations such as comparisons, arithmetic, string manipulation, etc.?

**IDEA**

Make built in functions into access paths with appropriate binding pattern.

**Example (Integer Inequalities)**

**Logical Schema:**  
\[
< /2, \leq /2 \subseteq S_L \text{ (written conventionally in infix)}
\]

**Physical Schema:**  
\[
\text{less}/2/2 \in S_A \\
\Rightarrow \Sigma_{LP} \supseteq \{ \forall x, y. (x < y) \leftrightarrow \text{less}(x, y) \} \\
\forall x, y. (x \leq y) \leftrightarrow \neg \text{less}(y, x) \}
\]

**Code:**

```python
function less-first
    return (x1 < x2)
function less-next
    return False
```

⇒ we already have `cmp/2/2` for equality!
Problem with Disks

Data is accessed in *blocks* (for efficiency)

⇒ NLJ accesses the *inner relation* number of tuples in the *outer relation*-times
Problem with Disks

Data is accessed in *blocks* (for efficiency)

⇒ NLJ accesses the *inner relation* number of tuples in the *outer relation*-times

Standard Solution: Block-based Operators

Block-NLJ operator:

1. read *as big block* of outer tuples in a memory buffer as possible
2. read a block from inner into a memory buffer
3. join the two buffers (producing output)
4. if inner not exhausted goto (2)
5. if outer not exhausted goto (1)
Two-level Store

Problem with Disks

Data is accessed in *blocks* (for efficiency)

⇒ NLJ accesses the *inner relation* number of tuples in the *outer relation*-times

Standard Solution: Block-based Operators

Block-NLJ operator:

1. read *as big block* of outer tuples in a memory buffer as possible
2. read a block from inner into a memory buffer
3. join the two buffers (producing output)
4. if inner not exhausted goto (2)
5. if outer not exhausted goto (1)

... is this extra code really necessary?
IDEA:
Split the access paths to a page reader and a record reader (that expects to be given a page already in memory).

Physical Schema:

$\Rightarrow S_A \supseteq \{\text{emp-pgscan}/1/0, \text{emp-recscan}/2/1\}$

$\Rightarrow \Sigma_{LP} \supseteq \{\forall x, y. (\text{emp-recscan}(y, x) \rightarrow \text{emp-pgscan}(y)), \forall x, y_1, y_2. ((\text{emp-recscan}(y_1, x) \land \text{emp-recscan}(y_2, x)) \rightarrow (y_1 \approx y_2)), \forall x. (\text{empfile}(x) \equiv \exists y. \text{emp-recscan}(y, x)) \}$
Query:

$$\exists y, z, w. (\text{employee}(x_1, y, z) \land \text{employee}(x_2, y, w))$$
Two-level Store Example

Query:

$$\exists y, z, w. (\text{employee}(x_1, y, z) \land \text{employee}(x_2, y, w))$$

Plan

E?y,?z,?w.(Employee(x1,?y,?z)^Employee(x2,?y,?w))
Plan: 803 (2n^2 + 50201,10000)
E?x6.(Emp-pgscan(?x6)^E?x4.(Emp-pgscan(?x4)^E?x5.(Emp-recscan(?x6,?x5)^Emp-num(?x5,x1)^E?x3.(Emp-recscan(?x4,?x3)^Emp-num(?x3,x2)^E?x2.(Emp-name(?x3,?x2)^E?s0.(Emp-name(?x5,?s0)^Cmp(?x2,?s0))))))))
Summary

1. Flexible modeling framework
   ⇒ new features = new access paths + constraints

2. Efficient query plans (comparable to hand-written code)