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

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

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
Main Memory and Pointers

Logical Schema:

```
employee
  number
  name
  salary

works
  enumber
  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).
Main Memory and Pointers: Formalization

**Logical Schema & Constraints:**

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

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

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

\[ \forall x, y, z. \text{department}(y, z, x) \rightarrow \exists u, v. \text{employee}(x, u, v), \ldots \} \]
Main Memory and Pointers: Formalization

Logical Schema & Constraints:
\[ S_L = \{ \text{employee}/3, \text{department}/3, \text{works}/2 \} ; \]
\[ \Sigma_L = \{ \forall x_1, x_2, y_1, y_2. \exists z. (\text{employee}(z, x_1, x_2) \land \text{employee}(z, y_1, y_2)) \rightarrow ((x_1 = y_1) \land (x_2 = y_2)), \]
\[ \forall x, y, z. (\text{works}(z, x) \land \text{works}(z, y)) \rightarrow (x = y), \]
\[ \forall x, y, z. \text{department}(y, z, x) \rightarrow \exists u, v. \text{employee}(x, u, v), \ldots \} \]

Physical Schema & Constraints:
\[ S_A = \{ \text{empfile}/1/0, \text{emp-num}/2/1, \]
\[ \text{emp-name}/2/1, \text{emp-salary}/2/1, \text{emp-dept}/2/1, \]
\[ \text{dept-num}/2/1, \text{dept-name}/2/1, \text{dept-manager}/2/1 \} , \]
\[ \Sigma_{LP} = \{ \forall x. (\text{empfile}(x) \rightarrow \exists y. \text{emp-num}(x, y)), \ldots, \]
\[ \forall x, y. (\text{emp-dept}(x, y) \rightarrow \text{deptfile}(y)), \]
\[ \forall x. (\text{deptfile}(x) \rightarrow \exists y. \text{dept-num}(x, y)), \ldots, \]
\[ \forall x, y. (\text{dept-manager}(x, y) \rightarrow \text{empfile}(y)), \]
\[ \forall x, y, z. (\text{employee}(x, y, z) \rightarrow \exists w. (\text{empfile}(w) \land \text{emp-num}(w, x))), \]
\[ \forall x, y, z, w. (\text{empfile}(w) \land \text{emp-num}(w, x) \land \text{emp-name}(w, y) \land \text{emp-salary}(w, z) \rightarrow \text{employee}(x, y, z)), \ldots \} \]
\( \exists z. \text{employee}(x, y, z) \):

E?z. Employee(x, y, ?z)

Plan: 26 (15n, 5n)

E?x1. (Empfile(?x1)^Emp-num(?x1, x)^Emp-name(?x1, y))
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)
   \[ E?x1. (\text{Empfile}(?x1) \land \text{Emp-num}(?x1, x) \land \text{Emp-name}(?x1, y)) \]

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

∃z.\text{employee}(x, y, z):
\begin{align*}
\text{E}\,?z.\text{Employee}(x, y, ?z) \\
\text{Plan: 26 (15n, 5n)}
\end{align*}
\begin{align*}
\text{E}\,?x1. (\text{Empfile}(?x1)^\text{Emp-num}(?x1, x)^\text{Emp-name}(?x1, y))
\end{align*}

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

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))

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))))

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

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

\textbf{Is there a shorter plan? YES:}
E?x2.(\texttt{Empfile}(?x2)^E?x1.(\texttt{Emp-dept}(?x2,?x1)^\texttt{Dept-name}(?x1,y)^\texttt{Dept-num}(?x1,x)^E?x3.\texttt{(Dept-manager}(?x1,?x3)^\texttt{Emp-num}(?x3,z)))

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

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

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

3. \( \exists y, v, w. \text{employee}(x_1, x_2, y) \land \text{works}(x_1, v) \land \text{department}(v, x_3, w) \):
   \[ \text{E?y, ?v, ?w.Employee}(x_1, x_2, ?y)^\text{Works}(x_1, ?v)^\text{Department}(?v, x_3, ?w) \]
   Plan: 50 (40n, 5n)
   \[ \text{E?x5.}(\text{Empfile}(?x5)^\text{Emp-num}(?x5,x_1)^\text{Emp-name}(?x5,x_2)^\text{E?x4.}(\text{Emp-dept}(?x5,?x4)^\text{Dept-name}(?x4,x_3))) \]
Hashing, Lists, et al.

Hashing with (list-based) Separate Chaining

Hash Array  Separate Chaining Linked Lists  DeptFile

Diagram showing the structure of Separate Chaining Linked Lists with nodes and arrows indicating connections.
Access paths:
⇒ \( \mathbb{S}_A \supseteq \{\text{hash}/2/1, \text{hasharraylookup}/2/1, \text{listscan}/2/1\} \).

Physical Constraints:
⇒ \( \sum_{LP} \supseteq \{\forall x, y.((\text{deptfile}(x) \land \text{dept-name}(x, y)) \rightarrow \exists z, w. (\text{hash}(y, z) \land \text{hasharraylookup}(z, w) \land \text{listscan}(w, x))), \forall x, y. (\text{hash}(x, y) \rightarrow \exists z. \text{hasharraylookup}(y, z)), \forall x, y. (\text{listscan}(x, y) \rightarrow \text{deptfile}(y))\} \).
Hashing, Lists, et al.

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

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

Queries:
\[ \exists y, z.(\text{department}(x_1, p, y) \land \text{employee}(y, x_2, z))\{p\}. \]

E?y,?z.Department(x1,p,?y)^Employee(?y,x2,?z) [p]
Plan: 497 (10,1)
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))))}
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:

\[
\frac{1}{2} < \frac{1}{2} \subseteq S
\]
(written conventionally in infix)

Physical Schema:

\[
\text{less} \left( \frac{1}{2}, \frac{1}{2} \right) \in S
\]

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

\[
\forall x, y. (x \leq y) \leftrightarrow \neg \text{less}(y, x)
\]

Code:

Function less-first

Function less-next

return (x1 < x2)

return false

⇒ we already have cmp \( / \) 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.
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: \(< /2, \leq /2 \subseteq S_L\) (written conventionally in infix)
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:**  
$\text{less}/2/2 \in S_A$  
$\Rightarrow \Sigma_{LP} \supseteq \{ \forall x, y. (x < y) \leftrightarrow \text{less}(x, y) \}$  
$\forall x, y. (x \leq y) \leftrightarrow \neg \text{less}(y, x) \}$
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:**  \(< \), \(\leq \), \(\subseteq S_L\) (written conventionally in infix)

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

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

**Code:**

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

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

function less-next
    return false
```

\(\Rightarrow \) we already have \(\text{cmp}/2/2\) for equality!
Two-level Store

Problem with Disks

Data is accessed in \textit{blocks} (for efficiency)
⇒ NLJ accesses the \textit{inner relation} number of tuples in the \textit{outer relation}-times

Standard Solution: Block-based Operators

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?
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:

\[
S_A \supseteq \{\text{emp-pgscan}/1/0, \text{emp-recscan}/2/1\}
\]

\[
\Sigma_{LP} \supseteq \{\forall x, y. (\text{emp-recscan}(y, x) \rightarrow \text{emp-pgscan}(y)),\]
\[
\forall x, y_1, y_2. ((\text{emp-recscan}(y_1, x) \land \text{emp-recscan}(y_2, x)) \rightarrow (y_1 \approx y_2)),
\]
\[
\forall x. (\text{empfile}(x) \equiv \exists y. \text{emp-recscan}(y, x)) \}
\]
Two-level Store Example

Query:

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

Query:

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

Plan

\[
\begin{align*}
E?y,?z,?w. & (\text{Employee}(x_1,?y,?z)^\text{Employee}(x_2,?y,?w)) \\
\text{Plan: 803} & (2n^2 + 50201,10000) \\
E?x6. & (\text{Emp-pgscan}(?x6)^\text{E?x4.}(\text{Emp-pgscan}(?x4)^{}) \\
E?x5. & (\text{Emp-recscan}(?x6,?x5)^\text{Emp-num}(?x5,x1)^{}) \\
E?x3. & (\text{Emp-recscan}(?x4,?x3)^\text{Emp-num}(?x3,x2)^{}) \\
E?x2. & (\text{Emp-name}(?x3,?x2)^E?s0.(\text{Emp-name}(?x5,?s0)^\text{Cmp}(?x2,?s0))))
\end{align*}
\]
Summary

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

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

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

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

Next time: HOW TO FIND REWRITINGS