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 *logically equivalent* to range-restricted queries over $\mathsf{S_A}$.
   - what does this kind of arrangement allow?
   - why is this *efficient*?
   - how to find such equivalent queries

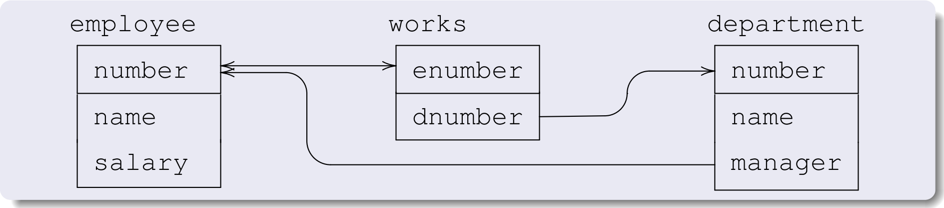
ADVANCED PHYSICAL DESIGNS

Case Studies

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

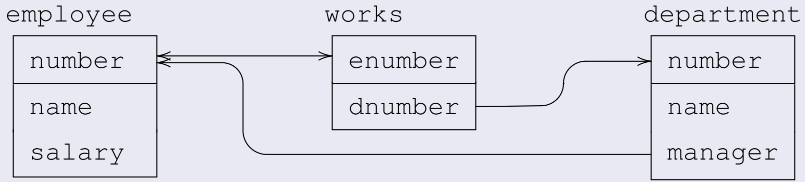
Main Memory and Pointers

Logical Schema:



Main Memory and Pointers

Logical Schema:



Physical Schema:

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

... and an array holding `emp` records (called `empfile`).

Main Memory and Pointers: Formalization

Logical Schema&Constraints:

- $\Rightarrow S_L = \{\texttt{employee}/3, \texttt{department}/3, \texttt{works}/2\};$
- $\Rightarrow \Sigma_L = \{\forall x_1, x_2, y_1, y_2.\exists z.(\texttt{employee}(z, x_1, x_2) \land \texttt{employee}(z, y_1, y_2)) \rightarrow ((x_1 = y_1) \land (x_2 = y_2)),$
  $\forall x, y, z.(\texttt{works}(z, x) \land \texttt{works}(z, y)) \rightarrow (x = y),$
  $\forall x, y, z.\texttt{department}(y, z, x) \rightarrow \exists u, v.\texttt{employee}(x, u, v), \ldots \}$

Main Memory and Pointers: Formalization

Logical Schema&Constraints:

- $\Rightarrow S_L = \{\texttt{employee}/3, \texttt{department}/3, \texttt{works}/2\};$
- $\Rightarrow \Sigma_L = \{\forall x_1, x_2, y_1, y_2.\exists z.(\texttt{employee}(z, x_1, x_2) \land \texttt{employee}(z, y_1, y_2)) \to ((x_1 = y_1) \land (x_2 = y_2)),$
  $\forall x, y, z.(\texttt{works}(z, x) \land \texttt{works}(z, y)) \to (x = y),$
  $\forall x, y, z.\texttt{department}(y, z, x) \to \exists u, v.\texttt{employee}(x, u, v), \ldots\}$

Physical Schema&Constraints:

- $\Rightarrow S_A = \{\texttt{empfile}/1/0, \texttt{emp-num}/2/1,$
  $\texttt{emp-name}/2/1, \texttt{emp-salary}/2/1, \texttt{emp-dept}/2/1,$
  $\texttt{dept-num}/2/1, \texttt{dept-name}/2/1, \texttt{dept-manager}/2/1\},$
- $\Rightarrow \Sigma_{LP} = \{\forall x.(\texttt{empfile}(x) \to \exists y.\texttt{emp-num}(x, y)), \ldots,$
  $\forall x, y.(\texttt{emp-dept}(x, y) \to \texttt{deptfile}(y)),$
  $\forall x.(\texttt{deptfile}(x) \to \exists y.\texttt{dept-num}(x, y)), \ldots,$
  $\forall x, y.(\texttt{dept-manager}(x, y) \to \texttt{empfile}(y)),$
  $\forall x.y, z.(\texttt{employee}(x, y, z) \to \exists w.(\texttt{empfile}(w) \land \texttt{emp-num}(w, x))),$
  $\forall x, y, z, w.((\texttt{empfile}(w) \land \texttt{emp-num}(w, x) \land \texttt{emp-name}(w, y) \land \texttt{emp-salary}(w, z)) \to \texttt{employee}(x, y, z)), \ldots\}$

Main Memory and Pointers: Queries and Plans

1. $\exists z.\mathtt{employee}(x, y, z)$:

```
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. $\exists z.\texttt{employee}(x, y, z)$:

   ```
   E?z.Employee(x,y,?z)
   Plan: 26 (15n,5n)
   E?x1.(Empfile(?x1)^Emp-num(?x1,x)^Emp-name(?x1,y))
   ```

2. $\texttt{Department}(x, y, z)$:

   ```
   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))))
   ```

Main Memory and Pointers: Queries and Plans

1. $\exists z.$`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?

Main Memory and Pointers: Queries and Plans

1. $\exists z.$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))
```

Main Memory and Pointers: Queries and Plans

1. $\exists z.$`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))
   ```

$\Rightarrow$ is it better?

Main Memory and Pointers: Queries and Plans

1. $\exists z.\texttt{employee}(x, y, z)$:

```
E?z.Employee(x,y,?z)
Plan: 26 (15n,5n)
E?x1.(Empfile(?x1)^Emp-num(?x1,x)^Emp-name(?x1,y))
```

2. $\texttt{Department}(x, y, z)$:

```
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))
```

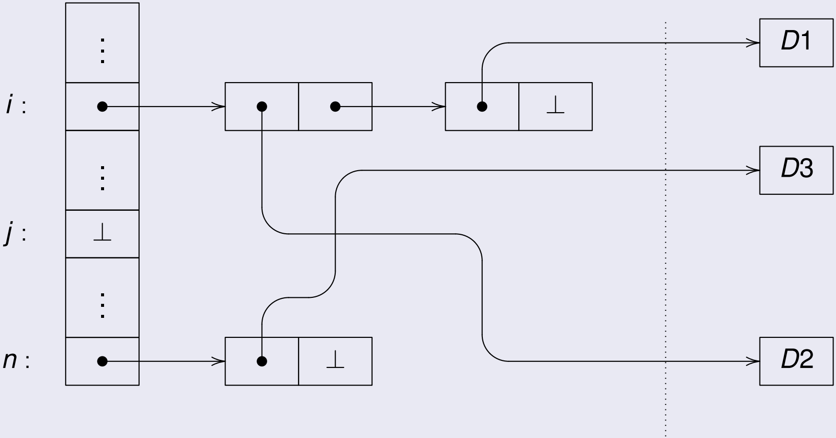
$\Rightarrow$ is it better? NO (duplicate elimination)

Main Memory and Pointers: Queries and Plans

1. $\exists z.\mathtt{employee}(x, y, z)$:
```
E?z.Employee(x,y,?z)
Plan: 26 (15n,5n)
E?x1.(Empfile(?x1)^Emp-num(?x1,x)^Emp-name(?x1,y))
```
2. $\mathtt{Department}(x, y, z)$:
```
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. $\exists y, v, w.\mathtt{employee}(x1, x2, y) \wedge \mathtt{works}(x1, v) \wedge \mathtt{department}(v, x3, w)$:
```
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)))
```

Hashing, Lists, et al.

Hashing with (list-based) Separate Chaining



Hash Array   Separate Chaining Linked Lists   DeptFile

Hashing, Lists, et al.

Access paths:

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

Physical Constraints:

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

University of Waterloo

Hashing, Lists, et al.

Access paths:

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

Physical Constraints:

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

Queries:

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

```
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.?

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)

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: $< /2, \leq /2 \subseteq \mathsf{S_L}$ (written conventionally in infix)

Physical Schema: $\texttt{less}/2/2 \in \mathsf{S_A}$

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

University of Waterloo

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)

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

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

Code:

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

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

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)

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

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

Code:

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

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

$\Rightarrow$ we already have $\texttt{cmp}/2/2$ for equality!

University of Waterloo

Two-level Store

Problem with Disks

Data is accessed in *blocks* (for efficiency)

$\Rightarrow$ NLJ accesses the *inner relation* number of tuples in the *outer relation*-times

Two-level Store

Problem with Disks

Data is accessed in *blocks* (for efficiency)

$\Rightarrow$ 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)

$\Rightarrow$ 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?

Two-level Store

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:

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

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

Two-level Store Example

Query:

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

Two-level Store Example

Query:

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

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
   $\Rightarrow$ new features = new access paths + constraints
2. Efficient query plans (comparable to hand-written code)

Summary

1. Flexible modeling framework
   $\Rightarrow$ new features = new access paths + constraints
2. Efficient query plans (comparable to hand-written code)

Next time: HOW TO FIND REWRITINGS