Fundamentals of Physical Design: Constraints and Indices

David Toman

D. R. Cheriton School of Computer Science
University of Waterloo
Recap of “State of Art”

- Current practice:
  Close coupling between Logical and Physical Schemata:
  \[ \Rightarrow \text{physical design} = \text{logical schema revision} + \text{indices} \]
  \[ \Rightarrow \text{makes query optimization “easy”} \]

  Logical schema revision \[ \Rightarrow \text{changes in application DML (BAD)} \]

- Alternative:
  Lose coupling supported by complex query optimization
  \[ \Rightarrow \text{must support a wide variety of physical designs} \]
DESIDERATA

Design a small number of **primitives** that support

- Conceptual/Logical schema development (including ICs)
- Physical schema development
- Linkage between the above two schemata

| 1 | uniform DDL for both conceptual/logical and physical objects |
| 2 | capabilities (index) declarations for physical objects |
| 3 | integrity constraints to establish links between objects |
| 4 | no built-in assumptions (e.g., 2-level store) |
Uniform Approach to Conceptual and Physical Design

DESIDERATA

Design a small number of primitives that support
- Conceptual/Logical schema development (including ICs)
- Physical schema development
- Linkage between the above two schemata

1. uniform DDL for both conceptual/logical and physical objects
2. capabilities (index) declarations for physical objects
3. integrity constraints to establish links between objects
4. no built-in assumptions (e.g., 2-level store)
Uniform Approach (cont.)

PLAN

The complete design is defined in terms of:

Integrity Constraints:

⇒ attach *attributes* to *classes/tables*
⇒ define *keys* and *foreign keys*
⇒ define *class hierarchies* (and *coverage*)
⇒ links conceptual and physical classes/tables

Index Declarations:

⇒ declare tables that can be *scanned* (binding patterns)
⇒ attaches *costs* to scanning these

... from now: a simple OO-style class/attribute based data model.
The complete design is defined in terms of:

**Integrity Constraints:**
- attach attributes to classes/tables
- define keys and foreign keys
- define class hierarchies (and coverage)
- links conceptual and physical classes/tables

**Index Declarations:**
- declare tables that can be scanned (binding patterns)
- attaches costs to scanning these

... from now: a simple OO-style class/attribute based data model.
Description Logic Syntax

Attributes and Path Functions: (denote total functions)

\[ \text{Pf} ::= \text{Id} \quad \text{identity} \quad \lambda x.x \]
\[ \mid \text{f.Pf} \quad \text{composition} \quad Pf \circ f \]

Concept Descriptions: (denote sets of objects)

\[ \text{C} ::= \text{A} \quad \text{primitive} \quad (A \subseteq \Delta) \]
\[ \mid \text{C}_1 \text{ and } \text{C}_2 \quad \text{intersection} \quad C_1 \cap C_2 \]
\[ \mid \text{not } \text{C} \quad \text{complement} \quad \Delta - C \]
\[ \mid \text{all Pf C} \quad \text{path type} \quad \{x \mid Pf(x) \in C\} \]
\[ \mid Pf_1 = Pf_2 \quad \text{equation} \quad \{x \mid Pf_1(x) = Pf_2(x)\} \]
\[ \mid \text{C: Pf}_1,\ldots,\text{Pf}_k \to \text{Pf} \quad \text{path FD} \]
\[ \{x \mid \forall y \in C. \bigwedge_{i=1}^{k} (Pf_i(x) = Pf_i(y)) \to (Pf(x) = Pf(y))\} \]

Constraints: \( \text{C}_1 < \text{C}_2 \) (denotes subset relation; schema = set of these)

\( \Rightarrow \) “\( \text{C}_1 < \text{C}_2 \)” a first order sentence: satisfiability, logical implication, \ldots
Integrity Constraints in Description Logic(s)

**Description Logic Syntax**

**Attributes and Path Functions:** (denote total functions)

\[ Pf ::= \text{Id} \quad \text{identity} \quad \lambda x.x \]
\[ \mid f. Pf \quad \text{composition} \quad Pf \circ f \]

**Concept Descriptions:** (denote sets of objects)

\[ C ::= A \quad \text{primitive} \quad (A \subseteq \Delta) \]
\[ \mid C_1 \text{ and } C_2 \quad \text{intersection} \quad C_1 \cap C_2 \]
\[ \mid \text{not } C \quad \text{complement} \quad \Delta - C \]
\[ \mid \text{all } Pf \ C \quad \text{path type} \quad \{ x \mid Pf(x) \in C \} \]
\[ \mid Pf_1 = Pf_2 \quad \text{equation} \quad \{ x \mid Pf_1(x) = Pf_2(x) \} \]
\[ \mid C : Pf_1, \ldots, Pf_k \rightarrow Pf \quad \text{path FD} \]
\[ \{ x \mid \forall y \in C. \bigwedge_{i=1}^{k} (Pf_i(x) = Pf_i(y)) \rightarrow (Pf(x) = Pf(y)) \} \]

**Constraints:** \( C_1 < C_2 \) (denotes *subset relation*; schema = set of these)

\( \Rightarrow \) “\( C_1 < C_2 \)” *a first order sentence*: satisfiability, logical implication, . . .
Example (Department and Employee Tables)

**EMPLOYEE** < (all Eid INT) and
(all Name STRING) and
(all Dept DEPARTMENT)

**DEPARTMENT** < (all City STRING) and
(all Boss EMPLOYEE)

Example (Department and Employee Keys)

**EMPLOYEE** < (EMPLOYEE: Eid -> Id)

**DEPARTMENT** < (DEPARTMENT: Boss.Eid -> Id)

**EMPLOYEE** < (not (Dept.Boss = Id))
DDL in DL Examples

Example (Department and Employee Tables)

EMPLOYEE < (all Eid INT) and  
(all Name STRING) and  
(all Dept DEPARTMENT)

DEPARTMENT < (all City STRING) and  
(all Boss EMPLOYEE)

Example (Department and Employee Keys)

EMPLOYEE < (EMPLOYEE: Eid -> Id)

DEPARTMENT < (DEPARTMENT: Boss.Eid -> Id)

EMPLOYEE < (not (Dept.Boss = Id))
Views via Integrity Constraints

Example (Employees Views)

\[
\text{WATEMP} < \text{EMPLOYEE} \text{ and } (\text{Dept.CITY} = \text{'Waterloo'}) \text{ and } \text{EMPLOYEE} < \text{WATEMP}
\]

\[
\text{TOKYOEMP} < \text{EMPLOYEE} \text{ and } (\text{Dept.CITY} = \text{'Tokyo'}) \text{ and } \text{WATEMP and TOKYOEMP} < \text{BOTTOM}
\]

Example (Coverage and Disjointness Constraints)

\[
\text{EMPLOYEE} < \text{WATEMP or TOKYOEMP}
\]

\[
\text{WATEMP and TOKYOEMP} < \text{BOTTOM}
\]
Views via Integrity Constraints

Example (Employees Views)

\[ \text{WATEMP} < \text{EMPLOYEE and } \]
\[ (\text{Dept.City} = \text{'Waterloo'}) \]
\[ \text{EMPLOYEE and } \]
\[ (\text{Dept.City} = \text{'Waterloo'}) < \text{WATEMP} \]
\[ \text{TOKYOEMP} < \text{EMPLOYEE and } \]
\[ (\text{Dept.City} = \text{'Tokyo'}) < \text{TOKYOEMP} \]

Example (Coverage and Disjointness Constraints)

\[ \text{EMPLOYEE} < \text{WATEMP or TOKYOEMP} \]
\[ \text{WATEMP and TOKYOEMP} < \text{BOTTOM} \]
IDEA: use generalized binding patterns

**A extra-logical declaration of the form**

\[
\text{index } A \ (Pf_1, \ldots, P_f m) \ (Pf_1', \ldots, Pf_n')
\]

where

- \( A \) is the (primitive) class whose objects are indexed,
- \( (Pf_1, \ldots, P_f m) \) are the *input parameters*, and
- \( (Pf_1', \ldots, Pf_n') \) are the *outputs*

in addition costs of getting the *first* and the *next* object

(details skipped in this presentation).

\[\Rightarrow\] each index declaration has an associated “iterator”.
Example: Addresses and Field Extraction

Example (Employee Table…)

EMPLOYEE < (all Eid INT) and
(all Name STRING) and
(all Dept DEPARTMENT)

Example (…as an array of pointers to structs)

EMPLOYEE < EARRAY < EMPLOYEE and (all Addr ADDR)
index EARRAY (Eid) (Addr)

EMPLOYEE < ENAME < EMPLOYEE
index ENAME (Addr) (Name)

EMPLOYEE < EDEPT < EMPLOYEE
index EDEPT (Addr) (Dept.Boss.Eid)
Example: 2-level Storage

Example (Department Table…)

DEPARTMENT < (all City STRING) and
(all Boss EMPLOYEE)

Example (…as a file of pages)

(all PgRef DPAGES) <
DEPARTMENT < DRECS < DEPARTMENT and
(all PgRef DPAGES) and (all Addr ADDR)
DPAGES < (all Addr ADDR)

index DPAGES () (Addr) ; expensive
index DRECS (PgRef.Addr) (Addr) ; cheap

⇒ now we can distinguish cost of “page access” v.s. “record access”
Example: Clustered/Unclustered Index Access

Example (Clustered index on Employee(Dept))

EMPLOYEE or (all PgRef CLUST) < EPAGE < EMPLOYEE and (all PgRef CLUST) and (Dept = PgRef.Dept)
index EPAGE (PgRef.Addr) (Addr)
index CLUST (Dept) (Addr)

Example (Un-Clustered index on Employee(Name))

EMPLOYEE or (all PgRef EPAGES) < EPAGE < EMPLOYEE and (all PgRef EPAGES) and (all CRef UNCLUST) and (PgRefPgId = Cref.PgId) and (Name = CRef.Name)
UNCLUST < (all PgId EPAGES)
index EPAGE (PgRef.Addr) (Addr)
index EPAGES (PgId) (Addr)
index UNCLUST (Name) (PgId)
Example: Denormalization

Example (EMPDEPT denormalization)

EMPDEPT < (all Eid INT) and (all Name STRING) and (all City STRING) (all Boss EMPDEPT)

EMPLOYEE < (all De EMPDEPT) and (EMPLOYEE: De->Id) and (Eid = De.Eid) and (Name = De.Name) and (Dept.City = De.City) and (Dept.Boss = De.Boss)

index EMPDEPT () (Eid,Name,City,Boss)

What happens to DEPT?

1. no additional info ⇒ we need a separate table (or NULLs)
2. every dept has an employee ⇒ additional constraints
Example: Denormalization

Example (EMPDEPT denormalization)

\[
\text{EMPDEPT} < (\text{all } \text{Eid INT}) \text{ and } (\text{all } \text{Name STRING}) \text{ and } \\
(\text{all } \text{City STRING}) \text{ and } (\text{all } \text{Boss EMPDEPT})
\]

\[
\text{EMPLOYEE} < (\text{all } \text{De EMPDEPT}) \text{ and } (\text{EMPLOYEE: De->Id}) \\
\text{and } (\text{Eid} = \text{De.Eid}) \text{ and } (\text{Name} = \text{De.Name}) \\
\text{and } (\text{Dept.City} = \text{De.City}) \\
\text{and } (\text{Dept.Boss} = \text{De.Boss})
\]

\[\text{index EMPDEPT () (Eid,Name,City,Boss)}\]

What happens to \text{DEPT}?

1. no additional info $\Rightarrow$ we need a separate table (or NULLs)
2. every dept has an employee $\Rightarrow$ additional constraints
Other Idioms

- Horizontal Partitioning
  ⇒ similar to \texttt{WATEMP-TOKYOEMP} example.

- Vertical Partitioning and (FK) Join Indices
  ⇒ similar to \textit{denormalization}

- Full Join Indices and Materialized Views
  ⇒ depends on the expressive power of the constraints: needs full FOL

- ...
On the Power of Integrity Constraints

Highly Expressive Logics

⇒ First-order Logic (algebraic dependencies)
   and extensions of FOL (fixpoints, . . .)
⇒ Logical Implication undecidable

Decidable Logics

⇒ (certain) Description Logics
⇒ Logical Implication decidable
   . . . to be combined with a decidable query language
⇒ Most features at modest cost

Weak Languages

⇒ status quo (∼ projections of “base relations”)
⇒ efficient but unable to cope with data independence
Summary

Take Home Message(s):

1. Integrity constraints *are key* to realizing the promise of *physical data independence*, and

2. Most of *physical design* issues (including appropriate costs) can be captured in such a framework.

To be solved . . .

How to *optimize queries*?

⇒ Constraints ~ (first-order) theories \((T)\)
⇒ Queries ~ (first-order) formulae \((Q)\)
⇒ Plans ~ (first-order) formulae (of certain shape, \(P\))

\[ T \models \forall \bar{x}. (Q \leftrightarrow P) \]

. . . additional issues: database trimmings (e.g., duplicates, etc.)
Summary

Take Home Message(s):

1. Integrity constraints are key to realizing the promise of physical data independence, and
2. Most of physical design issues (including appropriate costs) can be captured in such a framework.

To be solved . . .

How to optimize queries?

⇒ Constraints ∼ (first-order) theories ($T$)
⇒ Queries ∼ (first-order) formulae ($Q$)
⇒ Plans ∼ (first-order) formulae (of certain shape, $P$)

$T \models \forall \bar{x}.(Q \leftrightarrow P)$ .... but how do we find $P$???

... additional issues: database trimmings (e.g., duplicates, etc.)
Summary

Take Home Message(s):

1. Integrity constraints *are key* to realizing the promise of *physical data independence*, and
2. Most of *physical design* issues (including appropriate costs) can be captured in such a framework.

To be solved . . .

How to *optimize queries*?

⇒ Constraints \( \sim \) (first-order) theories (\( T \))
⇒ Queries \( \sim \) (first-order) formulae (\( Q \))
⇒ Plans \( \sim \) (first-order) formulae (of certain shape, \( P \))

\[ T \models \forall \bar{x}.(Q \leftrightarrow P) \]

. . . but how do we find \( P \)?

. . . additional issues: *database trimmings* (e.g., duplicates, etc.)
Summary

Take Home Message(s):

1. Integrity constraints are key to realizing the promise of physical data independence, and
2. Most of physical design issues (including appropriate costs) can be captured in such a framework.

To be solved . . .

How to optimize queries?

⇒ Constraints ∼ (first-order) theories ($T$)
⇒ Queries ∼ (first-order) formulae ($Q$)
⇒ Plans ∼ (first-order) formulae (of certain shape, $P$)

\[ T \models \forall \bar{x}.(Q \leftrightarrow P) \quad \ldots \text{ but how do we find } P???

. . . additional issues: database trimmings (e.g., duplicates, etc.)
Summary

Take Home Message(s):

1. Integrity constraints are key to realizing the promise of physical data independence, and
2. Most of physical design issues (including appropriate costs) can be captured in such a framework.

To be solved . . .

How to optimize queries?

⇒ Constraints ∼ (first-order) theories ($T$)
⇒ Queries ∼ (first-order) formulae ($Q$)
⇒ Plans ∼ (first-order) formulae (of certain shape, $P$)

$T \models \forall \bar{x}.(Q \leftrightarrow P)$ . . . but how do we find $P$???

. . . additional issues: database trimmings (e.g., duplicates, etc.)