Fundamentals of Physical Design
Open Problems

David Toman

D. R. Cheriton School of Computer Science
University of Waterloo
Take Home Massage(s):

1. **Basis for Physical Design**: expressive integrity constraints plus simple index (capabilities) declarations (plus cost estimates):
   - supports varied physical designs ranging from main-memory to external storage to distributed data.
   - provides a fine-grained control over how data is accessed using binding patterns.

2. **Query Optimization (compilation)**: essential part of the approach:
   - yields true physical data independence

3. Trade-offs between the expressive power of constraints/queries vs. the computational properties need to be considered.
Take Home Massage(s):

1. **Basis for Physical Design**: expressive *integrity constraints* plus simple *index (capabilities) declarations* (plus cost estimates):
   - supports varied *physical designs* ranging from main-memory to external storage to distributed data.
   - provides a *fine-grained* control over how data is accessed using *binding patterns*.

2. **Query Optimization (compilation)**: *essential* part of the approach:
   - yields true *physical data independence*

3. **Trade-offs** between the *expressive power* of constraints/queries vs. the *computational properties* need to be considered.
Take Home Massage(s):

1. Basis for *Physical Design*: expressive *integrity constraints* plus simple *index (capabilities) declarations* (plus cost estimates):
   - supports varied *physical designs* ranging from main-memory to external storage to distributed data.
   - provides a *fine-grained* control over how data is accessed using *binding patterns*.

2. *Query Optimization (compilation)*: *essential* part of the approach:
   - yields true *physical data independence*

3. Trade-offs between the *expressive power* of constraints/queries vs. the *computational properties* need to be considered.
Open Issues & Directions of Research

... for Interpolation:

1. Plan Generation and Costs
2. Duplicates, Binding Patterns, etc.

... for both/all Approaches:

3. Updates through Constraints
4. Ordering of Data
5. Inductive Types, Fixpoints, et al.
6. Transactions et al.
The interpolation based rewriting produces \textit{domain independent} query … from a proof of the \textit{implicit definability} property.

Can the above proof (search) be \textit{guided}:

\begin{enumerate}
\item to produce a \textit{range restricted} query instead? … and to respect \textit{binding patterns}?
\item to account for \textit{duplicate semantics}?
\item by the \textit{cost} (estimation) of the plan generated?
\end{enumerate}
The interpolation based rewriting produces *domain independent* query

... from a proof of the *implicit definability* property.

Can the above proof (search) be *guided*:

1. to produce a *range restricted* query instead?
   
   ... and to respect *binding patterns*?

2. to account for *duplicate semantics*?

3. by the *cost* (estimation) of the plan generated?
Updates through Constraints

Story so far:

1. Schema constraints
2. User Query

\[ \text{compile} \quad \rightarrow \quad \text{Query Plan} \]

Can this approach be extended to updates?

- how to specify what can change/what must remain invariant?
- how to deal with internal data values (e.g., page numbers)?
- how to handle non-determinism (e.g., page splits in B+tree)?
Updates through Constraints

Story so far:

1. Schema constraints
2. User Query

\[
\text{compile} \quad \rightarrow \quad \text{Query Plan}
\]

Can this approach be extended to *updates*?

- how to specify what can change/what must remain invariant?
- how to deal with *internal* data values (e.g., page numbers)?
- how to handle non-determinism (e.g., page splits in B+tree)?
Ordering of Data

Understanding *ordering* of data provides support for the use of *algorithmically better techniques* . . .

- removes the need for sorting (e.g., for duplicate removal)
- allows alternative algorithms (merge join, merge (union), etc.)

How to define proper semantics with *order*?
What are the appropriate physical primitives/operations?

Order Dependencies

Capture ordering correlations between attributes (paths):

\[ \text{EMP}<\text{EMP}: \text{Id}(\langle) \rightarrow \text{Eid}(\langle) \quad \text{Employees “ordered” by Eid} \]
\[ \text{EMP}<\text{EMP}: \text{Eid}(\langle) \rightarrow \text{Name}(\langle=) \quad \text{Names increase with Eids} \]
Ordering of Data

Understanding *ordering* of data provides support for the use of *algorithmically better techniques* . . .

- removes the need for sorting (e.g., for duplicate removal)
- allows alternative algorithms (merge join, merge (union), etc.)

- How to define proper semantics with *order*?
- What are the appropriate physical primitives/operations?

Order Dependencies

Capture ordering correlations between attributes (paths):

\[
\text{EMP} \prec \text{EMP}: \text{Id}(\prec) \rightarrow \text{Eid}(\prec) \quad \text{Employees “ordered” by Eid}
\]

\[
\text{EMP} \prec \text{EMP}: \text{Eid}(\prec) \rightarrow \text{Name}(\preceq) \quad \text{Names increase with Eids}
\]
Ordering of Data

Understanding *ordering* of data provides support for the use of *algorithmically better techniques* . . .

- removes the need for sorting (e.g., for duplicate removal)
- allows alternative algorithms (merge join, merge (union), etc.)

- How to define proper semantics with *order*?
- What are the appropriate physical primitives/operations?

Order Dependencies

Capture ordering correlations between attributes (paths):

\[
\text{EMP}<\text{EMP}: \text{Id}(\langle) \rightarrow \text{Eid}(\langle) \quad \text{Employees “ordered” by Eid} \\
\text{EMP}<\text{EMP}: \text{Eid}(\langle) \rightarrow \text{Name}(=\langle) \quad \text{Names increase with Eids}
\]
Inductive Types, Fixpoints, et al.

Physical Primitive in FO Approach
Index declarations + binding patterns (necessary to deal with sets)

Can we use *more primitive* constructs?

... only if queries/plans allow (some form of) iteration

- impact on schema language (e.g., regular expressions in Paths)?
- impact on query language (e.g., fixpoints, loops)?
  ... inductive types or general graphs?
- can we still *compile* queries?
Inductive Types, Fixpoints, et al.

Physical Primitive in FO Approach
Index declarations + binding patterns (necessary to deal with sets)

Can we use *more primitive* constructs?

... only if queries/plans allow (some form of) *iteration*

- impact on schema language (e.g., regular expressions in Paths)?
- impact on query language (e.g., fixpoints, loops)?
  ... inductive types or general graphs?
- can we still *compile queries*?
Transactions and Concurrency Control

IDEA

Describe *synchronization primitives* in the schema

... perhaps as a *special* index declaration

- can then queries/updates be compiled in such a way that they follow a particular *concurrency protocol* when executed?
  
  ... e.g., the *tree locking* protocol?

- how about *recovery*?
  
  ⇒ rollback for non-deadlock free CC?
  
  ⇒ durability?

D. Toman (Waterloo)
Transactions and Concurrency Control

### IDEA

Describe *synchronization primitives* in the schema

... perhaps as a *special* index declaration

- can then queries/updates be compiled in such a way that they follow a particular *concurrency protocol* when executed?
  
  ... e.g., the *tree locking* protocol?

- how about *recovery*?
  
  ⇒ rollback for non-deadlock free CC?
  ⇒ durability?