Logical Approach to Physical Data Independence and Query Compilation
Introduction, Background, and Goals

David Toman
D.R. Cheriton School of Computer Science
University of Waterloo
M/IMDB and Physical Data Independence

GOALS:

- Allow for use of *declarative* DML
  - such as SQL/OQL (or a fragment of SQL)
  - NoSQL is now “Not ONLY SQL”
- Allow for *low-level* data structures
  - navigating *pointers*
  - main-memory arrays
  - storage hierarchy
plans implement user requests. specifying physical design, expressing user requests and query plans, and understanding how query compilation. The methods are presented in terms of first-order logic which serves as the vehicle for to the fundamental methods underlying database technology that solves the problem of query compilation. Such capabilities relate to logical design is commonly called a physical design. This book is an introduction to a domain specific ways of understanding data, commonly called logical designs, to efficient executable concrete data sources, their interfaces and how to them. An appreciation of the

Query compilation is the problem of translating user requests formulated over purely conceptual and programs called query plans. Such plans access various concrete data sources through their low-level interfaces. An appreciation of the

USE SCENARIOS AND GOALS
IDEA:
Separate the users’ view(s) of the data from the way it is physically represented.

[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]
Physical Data Independence

IDEA:
Separate the users’ view(s) of the data from the way it is physically represented.

- independent customized user views,
- changes to conceptual structure without affecting users.

[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]
IDEA:
Separate the users’ view(s) of the data from the way it is physically represented.

- independent customized user views,
- changes to conceptual structure without affecting users,
- physical storage details hidden from users,
- changes to physical storage without affecting conceptual view,
Phyical Data Independence

IDEA:
Separate the users’ view(s) of the data from the way it is physically represented.

- physical storage details hidden from users,
- changes to physical storage without affecting conceptual view,

Originally just two levels: physical and conceptual/logical [Codd1970].

[ANSI/X3/SPARC Standards Planning and Requirements Committee, Bachman, 1975]
Example: PAYROLL

A Conceptual (user) view of PAYROLL data:

Example of PAYROLL data:
1. Mary is an employee.
2. Mary’s employee number is 3412.
3. Mary’s salary is 72000.

Example of PAYROLL:
4. There is a kind of entity called an employee.
5. There are attributes called enumber, name and salary.
6. Each employee entity has attributes enumber, name and salary.
7. Employees are identified by their enumber.
Example: PAYROLL

A physical design for PAYROLL:

There is a file of records called emp-file.
There are record fields emp-num, emp-name and emp-salary.
Each emp-file record has the fields emp-num, emp-name and emp-salary.
File emp-file is organized as a B-tree data structure that supports an emp-lookup operation, given a value for attribute enumber.

Records in file emp-file correspond one-to-one to employee entities.
Record fields in file emp-file encode the corresponding attribute values for employee entities, for example, emp-num encodes an enumber.
IDEA:

Queries are answered not only w.r.t. explicit data but also w.r.t. background knowledge

⇒ Ontology-based Data Access (OBDA)
IDEA:

Queries are answered not only w.r.t. *explicit data* but also w.r.t. *background knowledge*.

⇒ Ontology-based Data Access (OBDA)

Example

- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (background)

List all MORTALS ⇒ {Socrates} (query)
Ontology-based Data Access

IDEA:
Queries are answered not only w.r.t. *explicit data* but also w.r.t. *background knowledge* ⇒ Ontology-based Data Access (OBDA)

Example
- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (background)

List all MORTALs ⇒ {Socrates} (query)

[Calvanese et al.: Mastro]
Ontology-based Data Access

IDEA:
Queries are answered not only w.r.t. explicit data but also w.r.t. background knowledge
⇒ Ontology-based Data Access (OBDA)

Example
- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (background)

List all MORTALs ⇒ {Socrates} (query)

Question:
Is Aristoteles a MORTAL?
Ontology-based Data Access

IDEA:
Queries are answered not only w.r.t. explicit data but also w.r.t. background knowledge
⇒ Ontology-based Data Access (OBDA)

Example
- Socrates is a MAN (explicit data)
- Every MAN is MORTAL (background)

List all MORTALs ⇒ \{Socrates\} (query)

Question:
Is Aristoteles a MORTAL?

... can we really say “NO”?
PROBLEM:
How to transfer (reformat) data conforming to a *source schema* to data conforming to a *target schema*?

[Arenas et al: Foundations of Data Exchange]
PROBLEM:
How to transfer (reformat) data conforming to a source schema to data conforming to a target schema?

The general setting of data exchange is this:

source $S$ \(+\) mapping $M$ \(+\) target $T$ \(+\) query $Q$

[Arenas et al: Foundations of Data Exchange]
Data Exchange

**PROBLEM:**
How to transfer (reformat) data conforming to a *source schema* to data conforming to a *target schema*?

The general setting of data exchange is this:

![Diagram showing data exchange from source S to target T via mapping M and query Q]

[Arenas et al: Foundations of Data Exchange]

**Issues:**
- what should happen when the *target* is more complex than the *source*?
- how do we answer queries over the target?
IDEA:

Data integration provides a uniform access to a set of data sources, through a unified representation called global schema. A mapping specifies the relationship between the global schema and the sources.

[Genesereth: Data Integration]
IDEA:

Data integration provides a uniform access to a set of data sources, through a unified representation called global schema. A mapping specifies the relationship between the global schema and the sources.

[Genesereth: Data Integration]

Variants “which way do the arrows point” [Lenzerini]

GAV (global as a view), LAV (local as a view), and GLAV (“both ways”).
Common Threads and Issues

- In general two *schemas*: Conceptual/Logical and Physical
  - both endowed with *metadata* (vocabulary, ...)
  - mappings connect the schemas
  - (source) data only "in" the *physical* schema
  - queries only over the *conceptual/logical* schema

Issues to be formalized/fixed:
1. Formal description of the two schemas (same formalism for both?)
2. Language(s) for metadata and mappings
3. (user level) Data representation
4. (user level) Query language (semantics–aka when is an answer an answer?)
5. Algorithms/Execution model for queries: e.g., does materialization matter?
Common Threads and Issues

- In general two \textit{schemas}: Conceptual/Logical and Physical
  - both endowed with \textit{metadata} (vocabulary, \ldots)
  - mappings connect the schemas
  - (source) data only “in” the \textit{physical} schema
  - queries only over the \textit{conceptual/logical} schema

- Issues to be formalized/fixed:
  1. Formal description of the two schemas (same formalism for both?)
  2. Language(s) for metadata and mappings
  3. (user level) Data representation
  4. (user level) Query language (semantics—aka when is an answer an answer?)
Common Threads and Issues

- In general two *schemas*: Conceptual/Logical and Physical
  - both endowed with *metadata* (vocabulary, . . . )
  - mappings connect the schemas
  - (source) data only “in” the *physical* schema
  - queries only over the *conceptual/logical* schema

Issues to be formalized/fixed:
1. Formal description of the two schemas (same formalism for both?)
2. Language(s) for metadata and mappings
3. (user level) Data representation
4. (user level) Query language (semantics–aka when is an answer an answer?)
5. Algorithms/Execution model for queries: e.g., does *materialization* matter?
Phyical Data Independence: My Motivation

Goal: Application of the Ideas to Embedded Systems

1. High-level conceptual view of the system
2. High level query (and, eventually, update) language
3. Fine-grained physical schema description
4. Flexible conceptual-physical mappings
5. Queries (updates) compiled to operations on physical level

Challenge
The code generated from queries must be competitive with hand-written code.
Goal: Application of the Ideas to Embedded Systems

1. High-level conceptual view of the system [relational]
2. High level query (and, eventually, update) language [SQL]
3. Fine-grained physical schema description [records, pointers, ...]
4. Flexible conceptual-physical mappings
5. Queries (updates) compiled to operations on physical level [pointer navigation, field extraction, conditionals, ...]
Physical Data Independence: My Motivation

Goal: Application of the Ideas to Embedded Systems

1. High-level conceptual view of the system [relational]
2. High level query (and, eventually, update) language [SQL]
3. Fine-grained physical schema description [records, pointers, ...]
4. Flexible conceptual-physical mappings
5. Queries (updates) compiled to operations on physical level [pointer navigation, field extraction, conditionals, ...]

Challenge

The code generated from queries must be competitive with hand-written code.
Example of LINUX-INFO data:

1. process (called) gcc is running;
2. gcc's process number is 1234;
3. the user (id) running gcc is 145;
4. gcc uses files "foo.c" and "foo.o".
Example of LINUX-INFO data:
1. process (called) gcc is running;
2. gcc’s process number is 1234;
3. the user (id) running gcc is 145;
4. gcc uses files “foo.c” and “foo.o”.

Example of LINUX-INFO metadata:
5. There entities called process and file.
6. There are attributes called pno, pname, uname, and fname.
7. Each process entity has attributes pno, pname and uname.
8. Each file entity has attribute fname.
9. Processes are identified by their pno.
10. Files are identified by their fname.
11. There is a relationship uses between processes and files.
A physical design for Linux (selected by Linus Torvalds).

12 There are process records called task-struct.
13 Each task-struct record has record fields pid, uid, comm, and fds.
14 All task-structs is organized as a tree data structure.
15 The task-struct records correspond one-to-one to process entities.
16 Record fields in task-struct encode the corresponding attribute values for process entities, for example, pid encodes an pno, etc.
17 Similarly, fss correspond appropriately to (open) file entities.
18 fds field of task-struct is an array of fds; a non-null entry in this array indicates that the process corresponding to this task-struct is using the file identified by the name field of the fd record in the array.
User Query:

find all files used by processes invoked by user 145.
User Query:
find all files used by processes invoked by user 145.

Query Plan:

for each task-struct \( t \) in tree of task-structs
check if \( t \)'s uid field is 145 and, if so
scan the fds array in \( t \) and
if the file descriptor (fd) is non-NULL
print out the name of file field in fd.
User Query:

find all files used by processes invoked by user 145.

Query Plan:

for each task-struct \( t \) in tree of task-structs

check if \( t \)'s uid field is 145 and, if so

scan the fds array in \( t \) and

if the file descriptor (fd) is non-NULL

print out the name of file field in \( fd \).

Is the plan correct?

... and how do/can we answer this question?
Unifying Logic-based Approach
Vocabularies: Relational Model for both Conceptual and Physical Schemata.

Conceptual/Logical ($S_L$):

predicate symbols $R_1/a_1, \ldots, R_k/a_k$ ($a_i$ is the arity of $R_i$)
(possibly) constants $c_1, \ldots, c_n$
Vocabularies: Relational Model for both Conceptual and Physical Schemata.

Conceptual/Logical ($S_L$):

- predicate symbols $R_1/a_1, \ldots, R_k/a_k$ ($a_i$ is the arity of $R_i$)
- (possibly) constants $c_1, \ldots, c_n$

Physical ($S_P$):

- predicate symbols $S_1/b_1, \ldots, S_k/b_k$
- a distinguished subset $S_A \subseteq S_P$ of access paths
  - denote capabilities to retrieve tuples (i.e., data structures)
  - (optionally) binding patterns (restrictions on tuple retrieval)
  - associated with set of tuples (closed-world semantics)
Vocabularies: Relational Model for both Conceptual and Physical Schemata.

Conceptual/Logical ($S_L$):

- predicate symbols $R_1/a_1, \ldots, R_k/a_k$ ($a_i$ is the arity of $R_i$)
- (possibly) constants $c_1, \ldots, c_n$

Physical ($S_P$):

- predicate symbols $S_1/b_1, \ldots, S_k/b_k$
- a distinguished subset $S_A \subseteq S_P$ of access paths
  - denote capabilities to retrieve tuples (i.e., data structures)
  - (optionally) binding patterns (restrictions on tuple retrieval)
  - associated with set of tuples (closed-world semantics)

... a standard way of defining interpretations
Metadata and Constraints

Metadata: First-order sentences $\Sigma$ over $S_L \cup S_P$.

Conceptual/Logical ($\Sigma_L$):

$\Rightarrow$ keys, inclusion dependencies, hierarchies, . . .
Metadata and Constraints

Metadata: First-order sentences $\Sigma$ over $S_L \cup S_P$.

- Conceptual/Logical ($\Sigma_L$):
  - $\Rightarrow$ keys, inclusion dependencies, hierarchies, . . .

- Physical ($\Sigma_P$):
  - $\Rightarrow$ keys, inclusion dependencies, hierarchies, . . .
  - $\Rightarrow$ formulae that link to symbols in $S_L$ (mapping constraints).
Metadata and Constraints

Metadata: First-order sentences $\Sigma$ over $S_L \cup S_P$.

Conceptual/Logical ($\Sigma_L$):

$\Rightarrow$ keys, inclusion dependencies, hierarchies, . . .

Physical ($\Sigma_P$):

$\Rightarrow$ keys, inclusion dependencies, hierarchies, . . .
$\Rightarrow$ formulae that link to symbols in $S_L$ (mapping constraints).

. . . we resort to fragments of FOL to gain better computational properties
Example: LINUX-INFO

Conceptual/Logical:

\[ S_L = \{ \text{process}/3, \text{file}/1, \text{uses}/2 \} \]
\[ \Sigma_L = \{ \text{process}(x, y_1, z_1) \land \text{process}(x, y_2, z_2) \rightarrow y_1 = y_2 \land z_1 = z_2, \]
\[ \text{uses}(x, y) \rightarrow \exists z, w. \text{process}(x, z, w) \land \text{file}(y), \ldots \} \]

Physical:

\[ S_A = \{ \text{task struct}/1/0, \text{pid}/2/1, \text{uid}/2/1, \text{fds}/2/1, \text{fname}/2/1 \} \]
\[ \Sigma_P = \{ \text{task struct}(x) \rightarrow \exists y, z, w. \text{pid}(x, y) \land \text{uid}(z) \land \text{fds}(x, w) \]
\[ \text{pid}(x_1, y) \land \text{pid}(x_2, y) \rightarrow x_1 = x_1 \]
\[ \text{process}(x, y, z) \rightarrow \exists t. \text{task struct}(t) \land \text{pid}(t, x), \ldots \} \]
Queries: First-order formulae ($\varphi$) over $S_L$.

$$\Rightarrow \exists p, n, u.\text{process}(p, n, u) \land u = 145 \land \text{uses}(p, f) \land \text{file}(f)$$
Queries and Answers

Queries: First-order formulae ($\varphi$) over $S_L$.

$$\Rightarrow \exists p, n, u. \text{process}(p, n, u) \land u = 145 \land \text{uses}(p, f) \land \text{file}(f)$$

Data $D$:
Sets of (ground) tuples that fix meaning of every access path.
Queries and Answers

Queries: First-order formulae ($\varphi$) over $S_L$.

\[ \Rightarrow \exists p, n, u. \text{process}(p, n, u) \land u = 145 \land \text{uses}(p, f) \land \text{file}(f) \]

Data $D$:
Sets of (ground) tuples that fix meaning of every access path.

Query Answers:
answers \textit{in common} when evaluating $\varphi$ over \textit{every} interpretation (database) that is a model of $\Sigma$ and that extends $D$. 
Queries and Answers

Queries: First-order formulae ($\varphi$) over $S_L$.

$$\Rightarrow \exists p, n, u. \text{process}(p, n, u) \land u = 145 \land \text{uses}(p, f) \land \text{file}(f)$$

Data $D$:
Sets of (ground) tuples that fix meaning of every access path.

Query Answers:
answers *in common* when evaluating $\varphi$ over *every* interpretation (database) that is a model of $\Sigma$ and that extends $D$.

Definition (Certain Answers)

$$\text{cert}_{\Sigma, D}(\varphi) = \{ \bar{a} \mid \Sigma \cup D \models \varphi(\bar{a}) \}$$

logical implication

$$= \bigcap_{I \models \Sigma \cup D} \{ \bar{a} \mid I \models \varphi(\bar{a}) \}$$

answer in every model
The BAD News (and what can be done)

Theorem

“\( \bar{a} \in \text{cert}_{\Sigma,D}(\varphi) \)” is undecidable.

\[ \Rightarrow \] sources of undecidability: both \( \Sigma \) and \( \varphi \)!
The BAD News (and what can be done)

Theorem

“\( \vec{a} \in \text{cert}_{\Sigma,D}(\varphi) ? \)” is undecidable.

\[ \Rightarrow \text{sources of undecidability: both } \Sigma \text{ and } \varphi ! \]

Standard solution:

1. restrict \( \Sigma \) to decidable fragments of FOL (e.g., DLs)
2. restrict \( \varphi \) to a decidable fragment of FOL (e.g., UCQ)
The BAD News (and what can be done)

Theorem

✏️ \( \vec{a} \in \text{cert}_{\Sigma, D}(\varphi) \) is undecidable.

\[ \Rightarrow \] sources of undecidability: both \( \Sigma \) and \( \varphi \)!

Standard solution:

1. restrict \( \Sigma \) to decidable fragments of FOL (e.g., DLs)
2. restrict \( \varphi \) to a decidable fragment of FOL (e.g., UCQ)

<table>
<thead>
<tr>
<th>OBDA</th>
<th>( S_L, \Sigma_L )</th>
<th>( S_P, \Sigma_P )</th>
<th>queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Exchange</td>
<td>(lite) TBox</td>
<td>ABox</td>
<td>CQ/UCQ</td>
</tr>
<tr>
<td>Information Integration</td>
<td>target, target deps</td>
<td>source, st-tgds</td>
<td>CQ/UCQ</td>
</tr>
<tr>
<td></td>
<td>global view</td>
<td>local view, ( {G</td>
<td>L}AV )</td>
</tr>
</tbody>
</table>
IDEA: “make it look like a single model”

(severely) restrict what logical schema may look like:

every logical predicate $P(\vec{x})$ must correspond 1-1 to some access path.

... conceptual/logical symbols in queries are (mere aliases of) access paths.
... completely against the idea of physical data independence.
What do Relational Systems do??

**IDEA: “make it look like a single model”**

(severely) restrict what logical schema may look like:

> every logical predicate $P(\vec{x})$ must correspond 1-1 to some access path.

... conceptual/logical symbols in queries are (mere aliases of) access paths.

... completely against the idea of physical data independence.

Is this enough?
IDEA: “make it look like a single model”

(seriously) restrict what logical schema may look like:

every logical predicate $P(\vec{x})$ must correspond 1-1 to some access path.

... conceptual/logical symbols in queries are (mere aliases of) access paths.
... completely against the idea of physical data independence.

Is this enough? $\neg P(x)$? $\forall x. P(x)$?
What do Relational Systems do??

IDEA: “make it look like a single model”

(severely) restrict what logical schema may look like:

every logical predicate $P(\vec{x})$ must correspond 1-1 to some access path.

... conceptual/logical symbols in queries are (mere aliases of) access paths.
... completely against the idea of physical data independence.

Is this enough? $\neg P(x)$? $\forall x. P(x)$? ... depend on the domain of the model
What do Relational Systems do??

IDEA: “make it look like a single model”

(severely) restrict what logical schema may look like:

every logical predicate $P(\vec{x})$ must correspond 1-1 to some access path.

... conceptual/logical symbols in queries are (mere aliases of) access paths.
... completely against the idea of physical data independence.

IDEA-2: “only queries that think there is a single model”

A formula $\varphi$ is domain independent if for all pairs of models $I_1, I_2$ of $D$ and valuation $\theta$ we have

$$I_1, \theta \models \varphi \text{ if and only if } I_2, \theta \models \varphi.$$ 

... $I_1$ and $I_2$ can only differ in their domains (hence the name).
A LOGSPACE Algorithm

**IDEA**

Domain independent formulae can be evaluated in a model based on the *active domain of D* (set of individuals that appear in the access paths).
A LOGSPACE Algorithm

IDEA

Domain independent formulae can be evaluated in a model based on the active domain of $D$ (set of individuals that appear in the access paths).

... active domain of $D$ is a finite set.
A LOGSPACE Algorithm

IDEA
Domain independent formulae can be evaluated in a model based on the active domain of $D$ (set of individuals that appear in the access paths).

... active domain of $D$ is a finite set.

A Turing machine $T_\varphi$
- read only input tape storing (an encoding of) $\bar{a}$ and $D$;
- read/write work tape storing a counter for each variable in $\varphi$ (log $|D|$ bits) and fixed number of auxiliary counters;
- a finite control that implements top-down satisfaction check w.r.t. a valuation defined by the current state of the counters
  ⇒ used as pointers to individuals on the work tape.
A LOGSPACE Algorithm

IDEA

Domain independent formulae can be evaluated in a model based on the *active domain of D* (set of individuals that appear in the access paths).

... active domain of $D$ is a *finite set*.

A Turing machine $T_\varphi$

- read only input tape storing (an encoding of) $\vec{a}$ and $D$;
- read/write work tape storing a *counter* for each variable in $\varphi$ (log $|D|$ bits) and fixed number of auxiliary counters;
- a finite control that *implements* top-down satisfaction check w.r.t. a valuation defined by the current state of the counters
  $\Rightarrow$ used as pointers to individuals on the work tape.

Theorem

$$\text{cert}_{\Sigma, D}(\varphi) = \{\vec{a} \mid \langle \vec{a}, D \rangle \in \mathcal{L}(T_\varphi)\}.$$
Range-restricted Formulas and Relational Algebra

Nobody uses that algorithm!

Range-restricted Formulae (queries):
\[ \varphi ::= R(\vec{x}) \mid \varphi \land x = y \mid \varphi \land \varphi \mid \exists s. \varphi \mid \varphi \lor \varphi \mid \varphi \land \neg \varphi \]

Bottom-up “Algebraic” Query Evaluation:

every production above maps (at least naively) to an algebraic operation on finite relations:
- scan (with renaming),
- selection,
- join,
- projection,
- union, and
- difference.

Datalog (limited iteration)

additional predicates defined as a fixpoint positive query allows PTIME-complete problems.
Nobody uses that algorithm! Instead:

Range-restricted Formulae (queries):

\[ \varphi ::= R(\vec{x}) \mid \varphi \land x = y \mid \varphi \land \varphi \mid \exists s. \varphi \mid \varphi \lor \varphi \mid \varphi \land \neg \varphi \]

Bottom-up “Algebraic” Query Evaluation:

every production above maps (at least naively) to a algebraic operation on finite relations:

- scan (with renaming),
- selection,
- join,
- projection,
- union, and
- difference.
Range-restricted Formulas and Relational Algebra

Nobody uses that algorithm! Instead:

Range-restricted Formulae (queries):

\[ \varphi ::= R(\vec{x}) \mid \varphi \land x = y \mid \varphi \land \varphi \mid \exists s. \varphi \mid \varphi \lor \varphi \mid \varphi \land \neg \varphi \]

Bottom-up “Algebraic” Query Evaluation:

every production above maps (at least naively) to a algebraic operation on finite relations:

- scan (with renaming),
- selection,
- join,
- projection,
- union, and
- difference.

Datalog (limited iteration)

additional predicates defined as a fixpoint positive query allows PTIME-complete problems.
Summary

- comprehensive framework based on certain answers that unifies many database/KR approaches to handling information in presence of background information/theory/ontology;
- too expressive and in turn computationally in-feasible;
- practical (relational) systems: (almost) trivial instance of the framework.
Summary

- comprehensive framework based on certain answers that unifies many database/KR approaches to handling information in presence of background information/theory/ontology;
- too expressive and in turn computationally infeasible;
- practical (relational) systems: (almost) trivial instance of the framework.

Plan of Lectures:
1. Modeling Complex Physical Designs
2. Database Approach Extension and Interpolation
3. Classical OBDA: another way of gaining tractability (and its limits)
4. Updates of Data and Future Directions