

Logical Approach to Physical Data Independence and Query Compilation

Updates

David Toman

D.R. Cheriton School of Computer Science

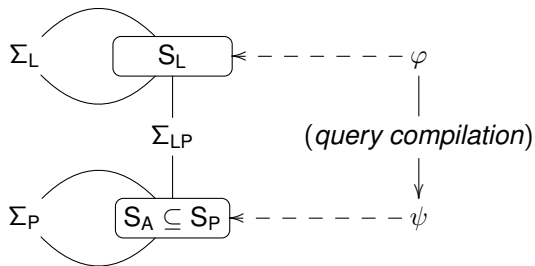
University of

Waterloo

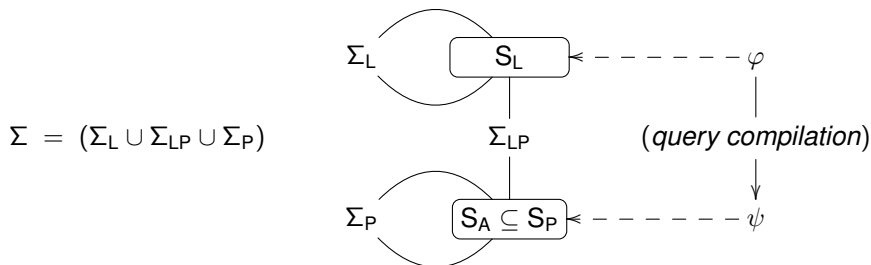


The Story So Far...

$$\Sigma = (\Sigma_L \cup \Sigma_{LP} \cup \Sigma_P)$$



The Story So Far...



Features:

- Flexible *physical design*: constraints $\Sigma_P \cup \Sigma_{LP}$ and code for S_A
⇒ main-memory operations, disk access, external sources of data, ... ;
- Query plans are efficient
⇒ all combination of *access paths* and simple operators;
⇒ *interpolation and postprocessing to find efficient ψ* .

DATA UPDATE

What is an Update?

User Update: $R_i \in S_L$

begin-transaction

$R_1^n := \varphi_{R_1}^o ;$

$R_2^n := \varphi_{R_2}^o ;$

\vdots

$R_m^n := \varphi_{R_m}^o ;$

end-transaction

What is an Update?

User Update: $R_i \in S_L$

begin-transaction

insert $\varphi_{R_1}^+$ into R_1 ; delete $\varphi_{R_1}^-$ from R_1 ;

insert $\varphi_{R_2}^+$ into R_2 ; delete $\varphi_{R_2}^-$ from R_2 ;

⋮

insert $\varphi_{R_m}^+$ into R_m ; delete $\varphi_{R_m}^-$ from R_m ;

end-transaction

What is an Update?

User Update: $R_i \in S_L$

```
begin-transaction
  insert  $\varphi_{R_1}^+$  into  $R_1$ ;    delete  $\varphi_{R_1}^-$  from  $R_1$ ;
  insert  $\varphi_{R_2}^+$  into  $R_2$ ;    delete  $\varphi_{R_2}^-$  from  $R_2$ ;
  :
  insert  $\varphi_{R_m}^+$  into  $R_m$ ;    delete  $\varphi_{R_m}^-$  from  $R_m$ ;
end-transaction
```

⇒ Assumption-1: transactions are *consistency-preserving*.

⇒ Assumption-2: results of $\varphi_{R_1}^+$ and $\varphi_{R_1}^-$ are given explicitly
as *finite sets of tuples*.

What is an Update?

User Update: $R_i \in S_L$

```
begin-transaction
  insert  $\varphi_{R_1}^+$  into  $R_1$ ;    delete  $\varphi_{R_1}^-$  from  $R_1$ ;
  insert  $\varphi_{R_2}^+$  into  $R_2$ ;    delete  $\varphi_{R_2}^-$  from  $R_2$ ;
  :
  insert  $\varphi_{R_m}^+$  into  $R_m$ ;    delete  $\varphi_{R_m}^-$  from  $R_m$ ;
end-transaction
```

⇒ Assumption-1: transactions are *consistency-preserving*.

⇒ Assumption-2: results of $\varphi_{R_i}^+$ and $\varphi_{R_i}^-$ are given explicitly
as *finite sets of tuples*.

Observation:

Typical *user transactions* modify only *few* relations

⇒ transaction types (specify which updates are *empty*).

What is the Problem?

- 1 how do we get from *user update* to *physical update*?
⇒ we need to synthesize φ_R^+, φ_R^- for each access path $R \in S_A$.

What is the Problem?

1 how do we get from *user update* to *physical update*?

⇒ we need to synthesize φ_R^+, φ_R^- for each access path $R \in S_A$.

... but what are the *available access paths* now?

What is the Problem?

- 1 how do we get from *user update* to *physical update*?
⇒ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in S_A$.
... but what are the *available access paths* now?
- 2 φ^+, φ^- may need *invented values* (RIDs): where do they come from?
⇒ *constant complement* idea (kind of an *oracle*).

What is the Problem?

- 1 how do we get from *user update* to *physical update*?
⇒ we need to synthesize φ_R^+, φ_R^- *for each access path* $R \in S_A$.
... but what are the *available access paths* now?
- 2 φ^+, φ^- may need *invented values* (RIDs): where do they come from?
⇒ *constant complement* idea (kind of an *oracle*).
- 3 access paths are modified *one-at-a-time*: what should the ordering be?
⇒ how does this affect the synthesis of φ^+, φ^- for other access paths?

UPDATING DATA VIA INTERPOLATION

Updating Access Paths

IDEA: “Update Schema” Σ^U

- 1 make two copies of the schema Σ :
 Σ^o (old: before update) and Σ^n (new: after update);

Updating Access Paths

IDEA: "Update Schema" Σ^U

- 1 make two copies of the schema Σ :
 Σ^o (old: before update) and Σ^n (new: after update);
- 2 add constraints describing the effect(s) of the *update*:

$$\Sigma^\pm = \left\{ \begin{array}{l} \forall x_1, \dots, x_k. R^+(x_1, \dots, x_k) \equiv R^n(x_1, \dots, x_k) \wedge \neg R^o(x_1, \dots, x_k), \\ \forall x_1, \dots, x_k. R^-(x_1, \dots, x_k) \equiv R^o(x_1, \dots, x_k) \wedge \neg R^n(x_1, \dots, x_k) \\ \quad \mid R/k \in S_L \cup S_P \}. \end{array} \right.$$

Updating Access Paths

IDEA: "Update Schema" Σ^U

- 1 make two copies of the schema Σ :
 Σ^o (old: before update) and Σ^n (new: after update);
- 2 add constraints describing the effect(s) of the *update*:

$$\Sigma^\pm = \left\{ \begin{array}{l} \forall x_1, \dots, x_k. R^+(x_1, \dots, x_k) \equiv R^n(x_1, \dots, x_k) \wedge \neg R^o(x_1, \dots, x_k), \\ \forall x_1, \dots, x_k. R^-(x_1, \dots, x_k) \equiv R^o(x_1, \dots, x_k) \wedge \neg R^n(x_1, \dots, x_k) \\ \quad \mid R/k \in S_L \cup S_P \}. \end{array} \right.$$

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in Σ^U ??

Updating Access Paths

IDEA: "Update Schema" Σ^U

- 1 make two copies of the schema Σ :
 Σ^o (old: before update) and Σ^n (new: after update);
- 2 add constraints describing the effect(s) of the *update*:

$$\Sigma^\pm = \left\{ \begin{array}{l} \forall x_1, \dots, x_k. R^+(x_1, \dots, x_k) \equiv R^n(x_1, \dots, x_k) \wedge \neg R^o(x_1, \dots, x_k), \\ \forall x_1, \dots, x_k. R^-(x_1, \dots, x_k) \equiv R^o(x_1, \dots, x_k) \wedge \neg R^n(x_1, \dots, x_k) \\ \quad \mid R/k \in S_L \cup S_P \}. \end{array} \right.$$

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in Σ^U ??

$$S_L = \{R^n \mid R \in S_A\} \cup \{R^+, R^- \mid R \in S_A\} \cup S_L^o \cup S_L^n;$$

$$S_A = \{R^o \mid R \in S_A\} \cup \{R^+, R^- \mid R/ \in S_L\}.$$

Updating Access Paths

IDEA: “Update Schema” Σ^U

- 1 make two copies of the schema Σ :
 Σ^o (old: before update) and Σ^n (new: after update);
- 2 add constraints describing the effect(s) of the *update*:

$$\Sigma^\pm = \left\{ \begin{array}{l} \forall x_1, \dots, x_k. R^+(x_1, \dots, x_k) \equiv R^n(x_1, \dots, x_k) \wedge \neg R^o(x_1, \dots, x_k), \\ \forall x_1, \dots, x_k. R^-(x_1, \dots, x_k) \equiv R^o(x_1, \dots, x_k) \wedge \neg R^n(x_1, \dots, x_k) \\ \quad \mid R/k \in S_L \cup S_P \}. \end{array} \right.$$

New schema $\Sigma^U = \Sigma^o \cup \Sigma^n \cup \Sigma^\pm$; logical symbols/access paths in Σ^U ??

$$S_L = \{R^n \mid R \in S_A\} \cup \{R^+, R^- \mid R \in S_A\} \cup S_L^o \cup S_L^n;$$

$$S_A = \{R^o \mid R \in S_A\} \cup \{R^+, R^- \mid R/ \in S_L\}.$$

Update code = interpolants for $R^+(\bar{x})$ and $R^-(\bar{x})$ under Σ^U (for each $R \in S_A$);
... still needs *code* that inserts/removes tuples into/from access paths.

Example

Relational Design: emp records *hold* dept-id (instead of pointer)

- User transaction of the form:

$$\{\text{employee}^+(123, \text{Bob}, \$50k), \text{works}^+(123, 345)\}$$

- Update code (for emp^+):

$$\text{emp}^+(x_1, x_2, x_3, x_4) := \text{employee}^+(x_1, x_2, x_3) \wedge \text{works}^+(x_1, x_4)$$

Example

Relational Design: emp records *hold* dept-id (instead of pointer)

- User transaction of the form:

$$\{\text{employee}^+(123, \text{Bob}, \$50k), \text{works}^+(123, 345)\}$$

- Update code (for emp^+):

$$\text{emp}^+(x_1, x_2, x_3, x_4) := \text{employee}^+(x_1, x_2, x_3) \wedge \text{works}^+(x_1, x_4)$$

... doesn't *quite* work with our *physical design with pointers* (why?)

Where do the “invented” Values come from?

Problem

- User transaction of the form:

$$\{\text{employee}^+(123, \text{Bob}, \$50k), \text{works}^+(123, 345)\}$$

- What should $\text{empfile}^+(w)$'s update code look like?

Where do the “invented” Values come from?

Problem

- User transaction of the form:

$$\{\text{employee}^+(123, \text{Bob}, \$50k), \text{works}^+(123, 345)\}$$

- What should $\text{empfile}^+(w)$'s update code look like?

$$\begin{aligned} &\forall x, y, z. (\text{employee}(x, y, z) \rightarrow \exists w. (\text{empfile}(w) \wedge \text{emp-num}(w, x))) \\ &\forall x, y, z, w. ((\text{employee}(x, y, z) \wedge \text{emp-num}(w, x)) \rightarrow \text{emp-name}(w, y)) \\ &\forall x, y, z, w, u. ((\text{employee}(x, y, z) \wedge \text{emp-num}(w, x) \wedge \text{works}(x, u) \\ &\quad \wedge \text{dept-num}(v, u)) \rightarrow \text{emp-dept}(w, v)). \end{aligned}$$

Where do the “invented” Values come from?

Problem

- What should $\text{empfile}^+(w)$'s update code look like?

$$\begin{aligned} &\forall x.y,z.(\text{employee}(x,y,z) \rightarrow \exists w.(\text{empfile}(w) \wedge \text{emp-num}(w,x))) \\ &\forall x,y,z,w.((\text{employee}(x,y,z) \wedge \text{emp-num}(w,x)) \rightarrow \text{emp-name}(w,y)) \\ &\forall x,y,z,w,u.((\text{employee}(x,y,z) \wedge \text{emp-num}(w,x) \wedge \text{works}(x,u) \\ &\quad \wedge \text{dept-num}(v,u)) \rightarrow \text{emp-dept}(w,v)). \end{aligned}$$

- 1 for $\text{empfile}^+(x)$:
$$\begin{aligned} &\exists y,z,t,u,v.\text{employee}^+(y,z,t) \wedge \text{works}^+(y,u) \\ &\quad \wedge \text{deptcomp}(u,v) \wedge \text{empcomp}(y,z,t,v,x) \end{aligned}$$
- 2 for emp-num , etc.: no-op.

Constant Complement Access Path

CC for emp records: empcomp/5/4

```
function empcomp-first
  if an emp record r with r->num = X1
    exists at address X5 return true
  X5 := new emp
  X5->num := X1
  X5->name := X2
  X5->sal := X3
  X5->dept := X4
  return true

function empcomp-next
  return false
```


Constant Complement Access Path

CC for emp records: empcomp/5/4

```
function empcomp-first
  if an emp record r with r->num = X1
    exists at address X5 return true

  X5 := new emp
  X5->num := X1
  X5->name := X2
  X5->sal := X3
  X5->dept := X4
  return true

function empcomp-next
  return false
```

Observation(s):

- “fills” all fields of a new record \Rightarrow emp-id, etc., no-ops;
- needs to check for existence of *all emp records* (not just in empfile!)

Schematic Cycles and Update Sequencing

Still a problem:

① for $\text{empfile}^+(x)$:

$$\begin{aligned} \exists y, z, t, u, v. & \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ & \wedge \text{dept comp}(u, v) \wedge \text{empcomp}(y, z, t, v, x), \end{aligned}$$

Schematic Cycles and Update Sequencing

Still a problem:

① for $\text{empfile}^+(x)$:

$$\begin{aligned} \exists y, z, t, u, v. & \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ & \wedge \text{deptcomp}(u, v) \wedge \text{empcomp}(y, z, t, v, x), \end{aligned}$$

... what should happen if department u doesn't exist?

Schematic Cycles and Update Sequencing

Still a problem:

① for $\text{empfile}^+(x)$:

$$\begin{aligned} \exists y, z, t, u, v. & \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ & \wedge \text{deptcomp}(u, v) \wedge \text{empcomp}(y, z, t, v, x), \end{aligned}$$

... what should happen if department u doesn't exist?

... ok, if it is a new department, who manages it?

Schematic Cycles and Update Sequencing

Still a problem:

① for $\text{empfile}^+(x)$:

$$\begin{aligned} &\exists y, z, t, u, v. \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ &\quad \wedge \text{deptcomp}(u, v) \wedge \text{empcomp}(y, z, t, v, x), \end{aligned}$$

... what should happen if department u doesn't exist?

... ok, if it is a new department, who manages it?

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
 \Rightarrow modify S_A as required when compiling AP update code.

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #1: force CC to use all attributes of the AP

Modify $\text{deptcomp}(y, x)$ to $\text{deptcomp}(y, n, m, x)/4/3$

⇒ i.e., force it to take complete dept records (same as empcomp).

① for $\text{empfile}^+(x)$:

$$\begin{aligned} \exists y, z, t, u, v. & \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ & \wedge \text{department}^+(u, n, m) \\ & \wedge \text{deptcomp}(u, n, x, v) \wedge \text{empcomp}(y, z, t, v, x) \end{aligned}$$

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #1: force CC to use all attributes of the AP

Modify $\text{deptcomp}(y, x)$ to $\text{deptcomp}(y, n, m, x)/4/3$

⇒ i.e., force it to take complete dept records (same as empcomp).

① for $\text{empfile}^+(x)$:

$$\begin{aligned} \exists y, z, t, u, v. & \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u) \\ & \wedge \text{department}^+(u, n, m) \\ & \wedge \text{deptcomp}(u, n, x, v) \wedge \text{empcomp}(y, z, t, v, x) \end{aligned}$$

How do you insert the first employee and department??

OOPS: not definable (because of *binding patterns*)

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

① for $\text{deptfile}^+(x)$:
 $\exists y, z, t. \text{department}^+(z, y, t) \wedge \text{deptcomp}(z, x),$

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

- 1 for $\text{deptfile}^+(x)$:
 $\exists y, z, t. \text{department}^+(z, y, t) \wedge \text{deptcomp}(z, x),$
- 2 for $\text{empfile}^+(x)$:
 $\exists y, z, t, u, v. \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u)$
 $\wedge \text{deptfile}(v) \wedge \text{dept-num}(v, u) \wedge \text{empcomp}(y, z, t, v, x),$

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

- 1 for $\text{deptfile}^+(x)$:
 $\exists y, z, t. \text{department}^+(z, y, t) \wedge \text{deptcomp}(z, x),$
- 2 for $\text{empfile}^+(x)$:
 $\exists y, z, t, u, v. \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u)$
 $\wedge \text{deptfile}(v) \wedge \text{dept-num}(v, u) \wedge \text{empcomp}(y, z, t, v, x),$
- 3 for $\text{dept-manager}^+(x, y)$:
 $\exists z, t, u. \text{department}^+(z, t, u) \wedge \text{deptfile}(x) \wedge \text{dept-num}(x, z)$
 $\wedge (\exists z, t, v, w. \text{employee}^+(u, z, t) \wedge \text{works}^+(u, v)$
 $\wedge \text{empfile}(y) \wedge \text{emp-id}(u, y))$

Schematic Cycles and Update Sequencing

IDEA

Constant Complement can ONLY be used with the AP that *stores* the records
⇒ modify S_A as required when compiling AP update code.

Attempt #2: stage updates via reification of attributes

- 1 for $\text{deptfile}^+(x)$:
 $\exists y, z, t. \text{department}^+(z, y, t) \wedge \text{deptcomp}(z, x),$
- 2 for $\text{empfile}^+(x)$:
 $\exists y, z, t, u, v. \text{employee}^+(y, z, t) \wedge \text{works}^+(y, u)$
 $\wedge \text{deptfile}(v) \wedge \text{dept-num}(v, u) \wedge \text{empcomp}(y, z, t, v, x),$
- 3 for $\text{dept-manager}^+(x, y)$:
 $\exists z, t, u. \text{department}^+(z, t, u) \wedge \text{deptfile}(x) \wedge \text{dept-num}(x, z)$
 $\wedge (\exists z, t, v, w. \text{employee}^+(u, z, t) \wedge \text{works}^+(u, v)$
 $\wedge \text{empfile}(y) \wedge \text{emp-id}(u, y))$
- 4 for $\text{dept-name}^+(x, y)$:
 $\exists z, t. \text{department}^+(z, y, t) \wedge \text{deptfile}(x) \wedge \text{dept-num}(x, z)$

Summary

- code for update of an access path
 - 1 synthesized queries ψ^+, ψ^- over *update schema*,
 - 2 code for primitive *inserts/deletes*,
 - 3 code for *constant complement* access paths (for “invented values”);
- schematic cycles must be broken via reification;

Summary

- code for update of an access path
 - 1 synthesized queries ψ^+, ψ^- over *update schema*,
 - 2 code for primitive *inserts/deletes*,
 - 3 code for *constant complement* access paths (for “invented values”);
- schematic cycles must be broken via reification;
- not entirely satisfactory (e.g., no in-place update)