Topics in Database Systems:
Main/in-memory and Embedded DBMS
CS848 Spring 2018

David Toman

DATABASE IMPLEMENTATION
(POSTPROCESSING AND EFFICIENCY)
BEYOND FO OPERATIONS
Aditional/Modified Operations

Why?

⇒ to improve *computational behaviour* of plans

1. splitting operations:
   - projection to *duplicate preserving projection and duplicate elimination* ({.})
   - union to *duplicate preserving union and duplicate elimination*
   
   . . . with the intention of *eventually eliminating* duplicate elimination

2. adding operations
   - cut and cut insertion
   
   . . . with the intention of *eliminating unsatisfiable searches*

3. improving behaviour of existing operations (access paths)
   
   . . . with the intention of *improving algorithms using ordered access*
Aditional/Modified Operations

Why?
⇒ to improve *computational behaviour* of plans

1. splitting operations:
   - projection to *duplicate preserving projection* and *duplicate elimination* (\(\{}\).\(\})
   - union to *duplicate preserving union* and *duplicate elimination*
     . . . with the intention of *eventually eliminating* duplicate elimination

2. adding operations
   - cut and cut insertion
     . . . with the intention of *eliminating unsatisfiable searches*

3. improving behaviour of existing operations (access paths)
   . . . with the intention of *improving algorithms using ordered access*
Adiitional/Modified Operations

Why?

⇒ to improve *computational behaviour* of plans

1. splitting operations:
   - projection to *duplicate preserving projection* and *duplicate elimination* (\{.\})
   - union to *duplicate preserving union* and *duplicate elimination*
     . . . with the intention of *eventually eliminating* duplicate elimination

2. adding operations
   - cut and cut insertion
     . . . with the intention of *eliminating unsatisfiable searches*

3. improving behaviour of existing operations (access paths)
   . . . with the intention of *improving algorithms using ordered access*
Additional/Modified Operations

Why?

⇒ to improve *computational behaviour* of plans

1. splitting operations:
   - projection to *duplicate preserving projection* and *duplicate elimination* (\{.,\})
   - union to *duplicate preserving union* and *duplicate elimination*
     . . . with the intention of *eventually eliminating* duplicate elimination

2. adding operations
   - cut and cut insertion
     . . . with the intention of *eliminating unsatisfiable searches*

3. improving behaviour of existing operations (access paths)
   . . . with the intention of *improving algorithms using ordered access*
Duplicates and Post-processing
Eliminating Duplicate Elimination

Consider a query plan $Q'$ obtained from a user query $Q$.

$Q'$ will in general require a top-level duplicate elimination operation to ensure that the query plan implements $Q$.

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design, and that $Q_1$ and $Q_2$ are a pair of query plans over the design.

A rewrite rule is written as $Q_1 \leftrightarrow Q_2$ and is correct if the following holds for all user queries $Q$ over the design.

$$\Sigma \models Q \triangleleft Q_1 \text{ iff } \Sigma \models Q \triangleleft Q_2$$
Eliminating Duplicate Elimination

Consider a query plan $Q'$ obtained from a user query $Q$.

$Q'$ will in general require a top-level *duplicate elimination* operation to ensure that the query plan implements $Q$.

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume $(S_L \cup S_P, \Sigma)$ is a physical design, and that $Q_1$ and $Q_2$ are a pair of query plans over the design.

A *rewrite rule* is written as $Q_1 \leftrightarrow Q_2$ and is correct if the following holds for all user queries $Q$ over the design.

$$\Sigma \models Q \models Q_1 \quad \text{iff} \quad \Sigma \models Q \models Q_2$$
Eliminating Duplicate Elimination

Consider a query plan $Q'$ obtained from a user query $Q$.

$Q'$ will in general require a top-level duplicate elimination operation to ensure that the query plan implements $Q$.

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume $\langle SL \cup SP, \Sigma \rangle$ is a physical design, and that $Q_1$ and $Q_2$ are a pair of query plans over the design.

A rewrite rule is written as $Q_1 \leftrightarrow Q_2$ and is correct if the following holds for all user queries $Q$ over the design.

$$\Sigma \models Q \triangleleft Q_1 \text{ iff } \Sigma \models Q \triangleleft Q_2$$
Eliminating Duplicate Elimination

Consider a query plan $Q'$ obtained from a user query $Q$.

$Q'$ will in general require a top-level *duplicate elimination* operation to ensure that the query plan implements $Q$.

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design, and that $Q_1$ and $Q_2$ are a pair of query plans over the design.

A *rewrite rule* is written as $Q_1 \leftrightarrow Q_2$ and is correct if the following holds for all user queries $Q$ over the design.

\[
\Sigma \models Q \triangleleft Q_1 \iff \Sigma \models Q \triangleleft Q_2
\]
Eliminating Duplicate Elimination

Consider a query plan \( Q' \) obtained from a user query \( Q \).

\( Q' \) will in general require a top-level *duplicate elimination* operation to ensure that the query plan implements \( Q \).

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume \( \langle S_L \cup S_P, \Sigma \rangle \) is a physical design, and that \( Q_1 \) and \( Q_2 \) are a pair of query plans over the design.

A *rewrite rule* is written as \( Q_1 \leftrightarrow Q_2 \) and is correct if the following holds for all user queries \( Q \) over the design.

\[
\Sigma \models Q \triangleleft Q_1 \ \text{iff} \ \Sigma \models Q \triangleleft Q_2
\]
Eliminating Duplicate Elimination

Consider a query plan $Q'$ obtained from a user query $Q$.

$Q'$ will in general require a top-level *duplicate elimination* operation to ensure that the query plan implements $Q$.

Adding duplicate elimination unconditionally to any query plan is clearly unacceptable on performance grounds. We now consider how to rewrite query plans to reduce and possibly avoid the overhead that this entails.

Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design, and that $Q_1$ and $Q_2$ are a pair of query plans over the design.

A *rewrite rule* is written as $Q_1 \leftrightarrow Q_2$ and is correct if the following holds for all user queries $Q$ over the design.

$$\Sigma \models Q \triangleleft Q_1 \quad \text{iff} \quad \Sigma \models Q \triangleleft Q_2$$
Query Context

Assume $Q_1$ is a query plan that contains a subplan $Q_2$. Write $Q^c_1$ to denote a query context in which $Q_2$ has been replaced by a placeholder “[]”. Also write $Q^c_1[Q']$ to denote a substitution of the placeholder “[]” in query context $Q^c_1$ with $Q'$ (either a query or a query context).

Observation: Contexts can be composed: if $Q^c_1$ and $Q^c_2$ are contexts, then $Q^c_1[Q^c_2]$ is a context.

Given a context $Q^c$, a user query $Uq_p(Q^c)$ abstracting properties of variables within the context is defined as follows.

$$Uq_p(Q^c) = \begin{cases} 
\top & Q^c = "[]" \\
Uq(Q_2) \land Uq_p(Q^c_1) & Q^c = "Q^c_1[Q_2 \land []]" \text{ or } "Q^c_1[[] \land Q_2]" \\
\exists x. Uq_p(Q^c_1) & Q^c = "Q^c_1[\exists x.[]]" \\
Uq_p(Q^c_1) & Q^c = "Q^c_1[\{[]\}]" \text{, } "Q^c_1[\neg []]" \text{, } "Q^c_1[Q_2 \lor []]" \text{ or } "Q^c_1[[] \lor Q_2]" 
\end{cases}$$
Query Context

Assume $Q_1$ is a query plan that contains a subplan $Q_2$. Write $Q_1^c$ to denote a query context in which $Q_2$ has been replace by a placeholder “[]”. Also write $Q^c[Q']$ to denote a substitution of the placeholder “[]” in query context $Q^c$ with $Q'$ (either a query or a query context).

Observation: Contexts can be composed: if $Q_1^c$ and $Q_2^c$ are contexts, then $Q_1^c[Q_2^c]$ is a context.

Given a context $Q^c$, a user query $Uq_p(Q^c)$ abstracting properties of variables within the context is defined as follows.

$$Uq_p(Q^c) = \begin{cases} 
\top & Q^c = "[]" \\
Uq(Q_2) \land Uq_p(Q_1^c) & Q^c = "Q_1^c[Q_2 \land []]" \text{ or } "Q_1^c[[] \land Q_2]" \\
\exists x. Uq_p(Q_1^c) & Q^c = "Q_1^c[\exists x.[]]" \\
Uq_p(Q_1^c) & Q^c = "Q_1^c[[[]]]", "Q_1^c[\neg []]", "Q_1^c[Q_2 \lor []]" \\
\text{or } "Q_1^c[[] \lor Q_2]" & \end{cases}$$
Query Context

Assume $Q_1$ is a query plan that contains a subplan $Q_2$. Write $Q_1^c$ to denote a query context in which $Q_2$ has been replace by a placeholder “[]”. Also write $Q^c[Q']$ to denote a substitution of the placeholder “[]” in query context $Q^c$ with $Q'$ (either a query or a query context).

**Observation:** Contexts can be composed: if $Q_1^c$ and $Q_2^c$ are contexts, then $Q_1^c[Q_2^c]$ is a context.

Given a context $Q^c$, a user query $Uq_p(Q^c)$ abstracting properties of variables within the context is defined as follows.

$$Uq_p(Q^c) = \begin{cases} 
T & Q^c = "[]"

Uq(Q_2) \land Uq_p(Q_1^c) & Q^c = "Q_1^c[Q_2 \land []]" \text{ or } "Q_1^c[[\land Q_2]"

\exists x. Uq_p(Q_1^c) & Q^c = "Q_1^c[\exists x.[]]"

Uq_p(Q_1^c) & Q^c = "Q_1^c[[[]]]", "Q_1^c[[[]]]", "Q_1^c[Q_2 \lor []]"

\text{or } "Q_1^c[[] \lor Q_2]"
\end{cases}$$
Query Context

Assume $Q_1$ is a query plan that contains a subplan $Q_2$. Write $Q_1^c$ to denote a query context in which $Q_2$ has been replace by a placeholder “[]”. Also write $Q_1^c[Q']$ to denote a substitution of the placeholder “[]” in query context $Q_1^c$ with $Q'$ (either a query or a query context).

**Observation**: Contexts can be composed: if $Q_1^c$ and $Q_2^c$ are contexts, then $Q_1^c[Q_2^c]$ is a context.

Given a context $Q^c$, a user query $Uq_{\rho}(Q^c)$ abstracting properties of variables within the context is defined as follows.

$$Uq_{\rho}(Q^c) \equiv \begin{cases} 
\top & Q^c = "[]" \\
Uq(Q_2) \land Uq_{\rho}(Q_1^c) & Q^c = "Q_1^c[Q_2 \land []]" \text{ or } "Q_1^c[[] \land Q_2]" \\
\exists x. Uq_{\rho}(Q_1^c) & Q^c = "Q_1^c[\exists x. [[]]]" \\
Uq_{\rho}(Q_1^c) & Q^c = "Q_1^c[[]][]", "Q_1^c[[[]]]", "Q_1^c[Q_2 \lor []]" \text{ or } "Q_1^c[[] \lor Q_2]"
\end{cases}$$
Assume \( \langle S_L \cup S_P, \Sigma \rangle \) is a physical design and \( Q^c[Q'] \) a query plan. Then the following rewrite rules hold.

\[
\begin{align*}
Q^c[\{R(x_1, \ldots, x_k)\}] & \leftrightarrow Q^c[R(x_1, \ldots, x_k)] \\
Q^c[\{Q_1 \land Q_2\}] & \leftrightarrow Q^c[\{Q_1\} \land \{Q_2\}] \\
Q^c[\{\exists x.Q_1\}] & \leftrightarrow C_1 Q^c[\exists x.\{Q_1\}] \\
Q^c[\{\neg Q_1\}] & \leftrightarrow Q^c[\neg Q_1] \\
Q^c[\{\neg\{Q_1\}\}] & \leftrightarrow Q^c[\neg\{Q_1\}] \\
Q^c[\{Q_1 \lor Q_2\}] & \leftrightarrow C_2 Q^c[\{Q_1\} \lor \{Q_2\}] 
\end{align*}
\]

\(C_1\) and \(C_2\) correspond to the following respective conditions, where \(y_1\) and \(y_2\) in the former are fresh variable names not occurring in \(Q\) or \(Q_1\).

\[
\Sigma \cup \{Uq_p(Q^c) \land Uq(Q_1)[y_1/x] \land Uq(Q_1)[y_2/x]\} \models (y_1 \approx y_2)
\]

\[
\Sigma \cup \{Uq_p(Q^c)\} \models (Q_1 \land Q_2) \rightarrow \bot
\]
Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design and $Q^c[Q']$ a query plan. Then the following rewrite rules hold.

$$Q^c[\{R(x_1, \ldots, x_k)\}] \iff Q^c[R(x_1, \ldots, x_k)]$$
$$Q^c[\{Q_1 \land Q_2\}] \iff Q^c[\{Q_1\} \land \{Q_2\}]$$
$$Q^c[\{\exists x. Q_1\}] \iff Q^c[\exists x.\{Q_1\}]$$
$$Q^c[\{\neg Q_1\}] \iff Q^c[\neg Q_1]$$
$$Q^c[\{\neg \{Q_1\}\}] \iff Q^c[\neg \{Q_1\}]$$
$$Q^c[\{Q_1 \lor Q_2\}] \iff Q^c[\{Q_1\} \lor \{Q_2\}]$$

$C_1$ and $C_2$ correspond to the following respective conditions, where $y_1$ and $y_2$ in the former are fresh variable names not occurring in $Q$ or $Q_1$.

$$\Sigma \cup \{Uq_p(Q^c) \land Uq(Q_1)[y_1/x] \land Uq(Q_1)[y_2/x]\} \models (y_1 \approx y_2)$$
$$\Sigma \cup \{Uq_p(Q^c)\} \models (Q_1 \land Q_2) \rightarrow \bot$$
Incremental Query Context

Given a context $Q^c$, a *user query* $Uq_{ip}(Q^c)$ abstracting *incremental properties* of variables within the context is defined as follows.

$$Uq_{ip}(Q^c) \equiv \begin{cases} 
\top & Q^c = "[\]"
Uq(Q_2) \land Uq_{ip}(Q_1^c) & Q^c = "Q_1^c[Q_2 \land [\]]"
\exists x. Uq_{ip}(Q_1^c) & Q^c = "Q_1^c[\exists x. [\]]"
Uq_{ip}(Q_1^c) & Q^c = "Q_1^c[\{[\}]]"; "Q_1^c[\neg[\]]"; "Q_1^c[Q_2 \lor [\]]";
& "Q_1^c[[\] \lor Q_2]" or "Q_1^c[[\] \land Q_2]"
\end{cases}$$
Cut Insertion

Observe that the rewrite rules for duplicate elimination are bidirectional, and can therefore determine situations in which such operators can be added to a query plan.

This is useful when formulating additional rewrite rules that determine when cut operators can be inserted in query plans without any impact on their ability to implement user queries.

Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design and $Q^c[\{(Q_1) \land Q_2\}]$ a query plan. Then the following rewrite rule holds.

$$Q^c[\{(Q_1) \land Q_2\}] \leftrightarrow C Q^c[\{(Q_1)\}_c \land (Q_2 \land \ell)]$$

$C_1$ corresponds to the following condition, where $\text{Out}(Q_1) = \{x_1, \ldots, x_k\}$ and where each $y_i$ and $z_j$ are fresh variable names not occurring in $Q^c$, $Q_1$ or $Q_2$.

$$\Sigma \cup \{Uq_{ip}(Q^c) \land Uq((Q_1 \land Q_2)[y_1/x_1, \ldots, y_k/x_k]) \land Uq((Q_1 \land Q_2)[z_1/x_1, \ldots, z_k/x_k])\} \models (y_1 \approx z_1) \land \cdots \land (y_k \approx z_k)$$
Cut Insertion

Observe that the rewrite rules for duplicate elimination are bidirectional, and can therefore determine situations in which such operators can be *added* to a query plan.

This is useful when formulating additional rewrite rules that determine when cut operators can be inserted in query plans without any impact on their ability to implement user queries.

Assume \( (S_L \cup S_P, \Sigma) \) is a physical design and \( Q^c[(Q_1 \land Q_2)] \) a query plan. Then the following rewrite rule holds.

\[
Q^c[(Q_1 \land Q_2)] \iff Q^c[[Q_1]_\ell \land (Q_2 \land \ell)]
\]

\( C_1 \) corresponds to the following condition, where \( \text{Out}(Q_1) = \{x_1, \ldots, x_k\} \) and where each \( y_i \) and \( z_j \) are fresh variable names not occurring in \( Q^c, Q_1 \) or \( Q_2 \).

\[
\Sigma \cup \{Uq_{lp}(Q^c) \land Uq((Q_1 \land Q_2)[y_1/x_1, \ldots, y_k/x_k]) \land Uq((Q_1 \land Q_2)[z_1/x_1, \ldots, z_k/x_k])\} \\
= (y_1 \approx z_1) \land \cdots \land (y_k \approx z_k)
\]
Cut Insertion

Observe that the rewrite rules for duplicate elimination are bidirectional, and can therefore determine situations in which such operators can be added to a query plan.

This is useful when formulating additional rewrite rules that determine when cut operators can be inserted in query plans without any impact on their ability to implement user queries.

Assume \( \langle S_L \cup S_P, \Sigma \rangle \) is a physical design and \( Q^c[\{Q_1\} \land Q_2] \) a query plan. Then the following rewrite rule holds.

\[
Q^c[\{Q_1\} \land Q_2] \quad \overset{C_1}{\leftrightarrow} \quad Q^c[[\{Q_1\}]_{\ell} \land (Q_2 \land !\ell)]
\]

\( C_1 \) corresponds to the following condition, where \( \text{Out}(Q_1) = \{x_1, \ldots, x_k\} \) and where each \( y_i \) and \( z_j \) are fresh variable names not occurring in \( Q^c, Q_1 \) or \( Q_2 \).

\[
\Sigma \cup \{Uq_{ip}(Q^c) \land Uq((Q_1 \land Q_2)[y_1/x_1, \ldots, y_k/x_k]) \land Uq((Q_1 \land Q_2)[z_1/x_1, \ldots, z_k/x_k])\} \\
\models (y_1 \approx z_1) \land \cdots \land (y_k \approx z_k)
\]
Cut Insertion

Observe that the rewrite rules for duplicate elimination are bidirectional, and can therefore determine situations in which such operators can be added to a query plan.

This is useful when formulating additional rewrite rules that determine when cut operators can be inserted in query plans without any impact on their ability to implement user queries.

Assume $\langle S_L \cup S_P, \Sigma \rangle$ is a physical design and $Q^c[\{Q_1\} \land Q_2]$ a query plan. Then the following rewrite rule holds.

$$Q^c[\{Q_1\} \land Q_2] \leftrightarrow_{C_1} Q^c[[\{Q_1\}]_\ell \land (Q_2 \land \neg \ell)]$$

$C_1$ corresponds to the following condition, where $\text{Out}(Q_1) = \{x_1, \ldots, x_k\}$ and where each $y_i$ and $z_j$ are fresh variable names not occurring in $Q^c$, $Q_1$ or $Q_2$.

$$\Sigma \cup \{Uq_{ip}(Q^c) \land Uq((Q_1 \land Q_2)[y_1/x_1, \ldots, y_k/x_k]) \land Uq((Q_1 \land Q_2)[z_1/x_1, \ldots, z_k/x_k])\} \models (y_1 \approx z_1) \land \cdots \land (y_k \approx z_k)$$
Sorted Access
Can we have Merge-Join Style Processing (et al)?

IDEA:

- improve *ordered* access paths with *fingers*
  ⇒ modifies the behaviour of *get-first*
- use standard Nested Loops Join

How Well are we doing?

- seamlessly integrates with other operators (e.g., concatenation)
- pay-as-you-go behaviour: runs
- extension to two-level access to data
- ...
Can we have Merge-Join Style Processing (et al)?

**IDEA:**

- improve *ordered* access paths with *fingers*  
  ⇒ modifies the behaviour of *get-first*
- use standard Nested Loops Join

**How Well are we doing?**

- seamlessly integrates with other operators (e.g., concatenation)
- pay-as-you-go behaviour: runs
- extension to two-level access to data
- ...