3. Update Propagation
What is Update Propagation?

**Update propagation (UP)** is the process of „extrapolating“ changes of base relations to those derived relations depending on them.

The resulting *induced changes* are not actually performed (except for materialized views) but are purely virtual.

Nevertheless, it may be important to compute them!
Basic assumptions of our work on relational event monitoring:

- The specification of **logical events** is done via SQL views.

- In the presence of base table updates, the corresponding views turn into **continuous queries** (CQs).

- For efficiently evaluating CQs, they ought to be evaluated incrementally using **delta techniques** and update propagation!
Specialized Operator Trees (1)

How can update propagation be realized having specialized algebra equations?

original set of Datalog rules:

- **r₁:** \( p(X) \leftarrow t(X,Y), \text{not } q(X,Y) \)
- **r₂:** \( t(X,Y) \leftarrow s(X,Y), X>1 \)
- **r₃:** \( s(X,Y) \leftarrow b(X,Y) \)
- **r₄:** \( s(X,Y) \leftarrow c(X,Y) \)

corresponding RA operator tree:
Specialized Operator Trees (2)

set of incremental Datalog rules:

$$r_1: \quad p^+(X) \leftarrow t^+(X,Y), \text{not} \ q(X,Y)$$

$$\quad \cdots$$

$$r_2: \quad t^+(X,Y) \leftarrow s^+(X,Y), X>1$$

$$r_3: \quad s^+(X,Y) \leftarrow b^+(X,Y), \text{not} \ c(X,Y)$$

$$r_4: \quad s^+(X,Y) \leftarrow c^+(X,Y), \text{not} \ b(X,Y)$$

common sub expression $T^+$

corresponding RA operator tree:
How can update propagation be realized having a specialized operator tree?
Specialized Operator Trees (4)

How can update propagation be realized having a specialized operator tree?

1. **Push-Approach:**
   Any modification is directly pushed through the tree from bottom to top:

   - Any modification is directly pushed through the tree from bottom to top:

   - Any modification is directly pushed through the tree from bottom to top:

   - Any modification is directly pushed through the tree from bottom to top:

   - Any modification is directly pushed through the tree from bottom to top:

   - Any modification is directly pushed through the tree from bottom to top:
Specialized Operator Trees (5)

How can update propagation be realized having a specialized operator tree?

2. Pull-Approach:

View $T^+$ is queried in regular intervals for continuously determining changes w.r.t. $T$:
Update Propagation: Push vs. Pull

The push- and pull-approach are quite different at first glance:

**Push-Approach:**
- Every update is pushed so *every* possible induced update is determined.
- The propagation is **not goal-directed**, that is, even irrelevant induced updates are computed.

**Pull-Approach:**
- Solely relevant (‘pulled’) induced updates are computed, that is, the propagation is **goal-directed**.
- Depending on the chosen time interval for the pulling some induced updates might get lost (if no timestamped tuples are considered).
Motivating Example (1)

R:

\[
\begin{align*}
c(X) & \leftarrow s(X), \text{not } q(X). \\
s(X) & \leftarrow p(X,\_). \\
p(X,Y) & \leftarrow e(X,Y,\_). \\
h(X,Z) & \leftarrow e(X,\_,Z). \\
\end{align*}
\]

F:

\[
\begin{align*}
e(1,1,1). & \quad e(2,2,1). \quad q(2). \\
e(1,1,2). & \quad e(2,2,2). \quad q(3). \\
e(1,1,3). & \quad e(2,2,3). \\
\end{align*}
\]

F*:

\[
\begin{align*}
p(1,1). & \quad h(1,1). \quad h(2,1). \quad s(1). \\
p(2,2). & \quad h(1,2). \quad h(2,2). \quad s(2). \\
h(1,3). & \quad h(2,3). \quad c(1). \\
\end{align*}
\]

U_1:
{ + e(4,1,2) } !

U_2:
{ - e(2,2,2) } !

U_3:
{ + e(4,1,2) ; - e(2,2,2) } !
Motivating Example (2)

R:

\[ c(X) \leftarrow s(X), \text{not } q(X). \]
\[ s(X) \leftarrow p(X,\_). \]
\[ p(X,Y) \leftarrow e(X,Y,\_). \]
\[ h(X,Z) \leftarrow e(X,\_,Z). \]

Materialized (CQ)

\[
\begin{align*}
R: & \quad e(1,1,1). \quad e(2,2,1). \quad q(2). \\
 & \quad e(1,1,2). \quad e(2,2,2). \quad q(3). \\
 & \quad e(1,1,3). \quad e(2,2,3). \\
\end{align*}
\]

\[
\begin{align*}
F: & \quad p(1,1). \quad h(1,1). \quad h(2,1). \quad s(1). \\
 & \quad p(2,2). \quad h(1,2). \quad h(2,2). \quad s(2). \\
 & \quad h(1,3). \quad h(2,3). \quad c(1). \\
\end{align*}
\]

F*:

\[
\begin{align*}
\{ + e(4,1,2) ; - e(2,2,2) \} !
\end{align*}
\]

Intuitive computation of the follow-up changes induced on derived relations:

\[
\begin{align*}
+ p(4,1) \\
+ h(4,2) \\
- h(2,2)
\end{align*}
\]

\[
\begin{align*}
+ s(4) \\
+ c(4)
\end{align*}
\]

The computation of induced changes for h is useless here, because they have no effect on the changes of c.

Update Propagation has to be goal-directed!
UP for Integrity Checking (1)

\[R:\]
\[
\begin{align*}
e_{\text{cyclic}} &\leftarrow s(X). \\
s(X) &\leftarrow p(X, X). \\
p(X, Y) &\leftarrow e(X, Y). \\
p(X, Y) &\leftarrow e(X, Z), p(Z, Y).
\end{align*}
\]

\[I:\] constraint not e_{cyclic}.

\[F:\]
\[
\begin{align*}
e(1,2). \\
e(2,3). \\
e(3,4).
\end{align*}
\]

\[F^*:\]
\[
\begin{align*}
p(1,2). \\
p(1,3). \\
p(2,3). \\
p(2,4). \\
p(3,4). \\
p(1,4).
\end{align*}
\]

\[U_1: \{+ e(4,1) \} !
\]

\[U_2: \{- e(2,3) \} !
\]

\[U_3: \{+ e(4,1) ; - e(2,3) \} !
\]
UP for Integrity Checking (2)

R:

\[
\begin{align*}
e_{\text{cyclic}} & \leftarrow s(X). \\
s(X) & \leftarrow p(X,X). \\
p(X, Y) & \leftarrow e(X, Y). \\
p(X, Y) & \leftarrow e(X, Z), p(Z, Y).
\end{align*}
\]

I:

**constraint** not \( e_{\text{cyclic}}. \)

U_1:

\( \{ + e(4,1) \} ! \)

Intuitive computation of the follow-up changes induced on derived relations:

\[
\begin{align*}
+ p(4,1) \\
+ p(4,2) \\
+ p(4,3) \\
+ p(4,4)
\end{align*}
\]

\[
\begin{align*}
+ p(3,1) \\
+ p(3,2) \\
+ p(3,3)
\end{align*}
\]

\[
\begin{align*}
+ p(2,1) \\
+ p(2,2)
\end{align*}
\]

\[
\begin{align*}
+ p(1,1)
\end{align*}
\]

\[
\begin{align*}
+ s(1) \\
+ s(2) \\
+ s(3) \\
+ s(4)
\end{align*}
\]

\[
\begin{align*}
+ e_{\text{cyclic}}
\end{align*}
\]

\text{inconsistent}
UP for Integrity Checking (3)

R:

```
R:        E:      F:
  e_cyclic ← s(X).
  s(X) ← p(X,X).
  p(X,Y) ← e(X,Y).
  p(X,Y) ← e(X,Z), p(Z,Y).
```

I:

```
I:  constraint not e_cyclic.
```

U₂:

```
U₂:  {¬ e(2,3) } !
```

F:

```
F:  e(1,2).
    e(2,3).
    e(3,4).
```

F*:

```
F*:  p(1,2).  p(1,3).
    p(2,3).  p(2,4).
    p(3,4).  p(1,4).
```

No further induced changes

Strictly speaking, integrity checking is completely useless here, because an acyclic graph will never become cyclic by canceling an edge!
UP for Integrity Checking (4)

R:
\[
\begin{align*}
e_{\text{cyclic}} & \leftarrow s(X). \\
s(X) & \leftarrow p(X,X). \\
p(X,Y) & \leftarrow e(X,Y). \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y).
\end{align*}
\]

I:
constraint not e_{cyclic}.

U_3:
\[
\{ + e(4,1); - e(2,3) \}
\]

Combining both elementary updates in a single transaction does not lead to A combination of the induced effects, as there are side effects of the deletion compensating certain effects of the insertion:

F:
\[
\begin{align*}
e(1,2). \\
e(2,3). \\
e(3,4).
\end{align*}
\]

F*:
\[
\begin{align*}
p(1,2). & \ p(1,3). \\
p(2,3). & \ p(2,4). \\
p(3,4). & \ p(1,4).
\end{align*}
\]

Consistent, too, as there are no further induced changes.
Base Updates, Transitions, Induced Transitions

- **Base updates** are elementary changes (Insertions/Deletions) of individual facts in one base relation expressed by dynamic ground literals, e.g.:
  
  \[ +e(4,1,2) \]

- A **transition** \( \delta \) is a set of base updates which are employed to change the base relations of a database \( D \):

  \[ \{ +e(4,1,1), +e(4,1,2), -e(2,2,2), \ldots \} \]

- We assume that every transition solely contains **non-compensating** base updates; that is ground literals with opposite signs like \( +e(4) \) and \( -e(4) \) are not allowed.

- Let \( D \) be a database and \( \delta \) a transition from \( D^{\text{old}} \) to \( D^{\text{new}} \). Then \( \delta \) **induces a transition** \( \Delta \) from \( (F^{\text{old}})^* \) to \( (F^{\text{new}})^* \) which is a pair \( \langle \Delta^+, \Delta^- \rangle \) of sets of ground atoms (delta sets) such that:

  \[ \Delta^+ = (F^{\text{new}})^* \setminus (F^{\text{old}})^* \]  
  **Induced insertions**

  \[ \Delta^- = (F^{\text{old}})^* \setminus (F^{\text{new}})^* \]  
  **Induced deletions**
A transition $\delta$ from $F^{\text{old}}$ to $F^{\text{new}}$: $F^{\text{old}} = F$ and $F^{\text{new}} = \delta(F)$.
The induced transition comprises the complete effects of an update with respect to the base and derived relations:

\[
(F_{\text{old}})^* - (F_{\text{old}})^* \Delta (F_{\text{new}})^* - (F_{\text{new}})^* \delta (F)
\]

By comparing \((F_{\text{old}})^*\) and \((F_{\text{new}})^*\), the differences between both states can be determined. Doing so in practice would be a very naive way of computing induced updates!
Rule-Based UP: Principle (1)

Each individual update propagation step obviously is a special form of Forward Reasoning, where – in contrast with our previous T-operators – new changes are derived from facts and previous changes, e.g.:

\[
\begin{align*}
p(X,Y) & \leftarrow e(X,Z), \quad p(Z,Y). \\
+ & e(4,1) \\
p(1,2), \quad p(1,3), \quad p(2,3), \quad p(2,4), \quad p(3,4), \quad p(1,4). \\
Y \in \{2,3,4\}
\end{align*}
\]

Three phases of a single propagation step:

1. "Matching": U-literal matched with "suitable" body literal
2. Residual evaluation: Evaluation of the remaining body literals
3. Propagation: Transfer of the variable bindings resulting from 1 and 2 to the rule head and combination of the body literal with the sign of U
If summarizing all steps that are possible in a particular state of propagation into a set, a set-oriented iteration process can be discovered again, which continues until no further derivation is possible anymore:

Update propagation can be viewed as fixpoint iteration, too!
A Transformation-Based Approach to UP

This observation motivates an approach where automatically generated rules perform update propagation:

Rule Compiler

Internal Rules

per change pattern

Update

Induced updates and intermediate results
3.1 Intuitively Defining Update Rules
- Propagation Rules -
Update Propagation: Rule Compilation

Coding a single propagation step by means of a Datalog rule:

\[ p^+(X,Y) \leftarrow e^+(X,Z), \quad p(Z,Y). \]

Propagation rule

(or: Delta rule)

Internal auxiliary relations for temporarily storing information about base data updates and induced updates:

\[ (\Delta : \text{Difference between old and new state}) \]
Positive Propagation Rules

\[
\begin{align*}
  e_{\text{cyclic}} &\leftarrow s(X). \\
  s(X) &\leftarrow p(X,X). \\
  p(X,Y) &\leftarrow e(X,Y). \\
  p(X,Y) &\leftarrow e(X,Z), p(Z,Y).
\end{align*}
\]

\[
\begin{align*}
  e_{\text{cyclic}}^+ &\leftarrow s^+(X). \\
  s^+(X) &\leftarrow p^+(X,X). \\
  p^+(X,Y) &\leftarrow e^+(X,Y). \\
  p^+(X,Y) &\leftarrow e^+(X,Z), p^+(Z,Y). \\
  p^+(X,Y) &\leftarrow e^+(X,Z), p^+(Z,Y).
\end{align*}
\]

Each body literal may serve as starting point for propagation.

Propagating two changes simultaneously was not needed in our motivating example, but may be indispensable in other situations, e.g.:

\[
\begin{align*}
  U_4: & \{+e(4,5);+e(5,1)\}!
\end{align*}
\]
Positive Propagation Rules and FPI

\[ R \]

\[
\begin{align*}
e_{cyclic} & \leftarrow s(X). \\
s(X) & \leftarrow p(X,X). \\
p(X,Y) & \leftarrow e(X,Y). \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y).
\end{align*}
\]

\[ R^A_+ \]

\[
\begin{align*}
e_{cyclic}^+ & \leftarrow s^+(X). \\
s^+(X) & \leftarrow p^+(X,X). \\
p^+(X,Y) & \leftarrow e^+(X,Y). \\
p^+(X,Y) & \leftarrow e^+(X,Z), p^+(Z,Y). \\
p^+(X,Y) & \leftarrow e^+(X,Z), p^+(Z,Y).
\end{align*}
\]

\[ U_1: \]

\[
\{ + e(4,1) \} !
\]

Taking note of the change by means of adding a delta fact „triggering“ FPI: Seed fact !

\[
\begin{align*}
p^+(4,1) \\
p^+(4,2) \\
p^+(4,3) \\
p^+(4,4)
\end{align*}
\]

\[
\begin{align*}
p^+(3,1) \\
p^+(3,2) \\
p^+(3,3) \\
p^+(3,4) \\
s^+(4)
\end{align*}
\]

\[
\begin{align*}
p^+(2,1) \\
p^+(2,2) \\
p^+(2,3) \\
p^+(2,4) \\
s^+(3) \\
e_{cyclic}^+
\end{align*}
\]

\[
\begin{align*}
p^+(1,1) \\
p^+(1,2) \\
p^+(1,3) \\
p^+(1,4) \\
s^+(2)
\end{align*}
\]

\[ s^+(1) \]

Fixpoint reached!
Positive Propagation: Erroneous Derivations!

Comparing the "intuitively" determined difference with the result of fixpoint iteration over the delta rules exhibits a discrepancy: Some "old" paths are indicated as presumably new!
Analyzing the Mistake

- Some of the "wrong" delta facts have been generated due to already derivable p-facts being "rediscovered"! In these cases, an already existing derivation via the non-recursive rule has been confirmed by a **new derivation** induced via the recursive rule:

  - Already before: \( p(3,4) \leftarrow e(3,4). \)
  - Now in addition: \( p^+(3,4) \leftarrow e(3,4), p^+(4,4). \)

- Other "wrong" delta facts result from further "rediscoveries" arising as a consequence of the first mistake:

  - Before: \( p(2,4) \leftarrow e(2,3), p(3,4). \)
  - Now: \( p^+(2,4) \leftarrow e(2,3), p^+(3,4). \)
Avoiding Duplicates and Multiple Derivations

- In each of these cases, the source of error was a constellation where *duplicates* have been arising while computing (presumably new) derivations. In these cases, *multiple derivations* of the same fact occurred due to either union or projection:

- It is necessary to detect such multiple derivations already possible before the update, thus avoiding erroneous derivation of delta facts! This can be achieved by extending the delta rules by means of a test for derivability in the old state, which determines whether each new delta fact is effectively new:

- However, such an effectiveness test is required only if the respective relation is either defined by more than one rule (union), or if the defining rule contains local variables (projection).
Positive Prop. Rules with Effectiveness Test

\[ R \]

\[ e_{\text{cyclic}} \leftarrow s(X). \]
\[ s(X) \leftarrow p(X,X). \]
\[ p(X,Y) \leftarrow e(X,Y). \]
\[ p(X,Y) \leftarrow e(X,Z), p(Z,Y). \]

\[ R^{\Delta +} \]

\[ e_{\text{cyclic}}^+ \leftarrow s^+(X), \text{not } e_{\text{cyclic}}. \]
\[ s^+(X) \leftarrow p^+(X,X), \text{not } s(X). \]
\[ p^+(X,Y) \leftarrow e^+(X,Y), \text{not } p(X,Y). \]
\[ p^+(X,Y) \leftarrow e^+(X,Z), p(Z,Y), \text{not } p(X,Y). \]
\[ p^+(X,Y) \leftarrow e^+(X,Z), p^+(Z,Y), \text{not } p(X,Y). \]

\[ U_1: \{ + e(4,1) \}! \]

With the effectiveness test, FPI delivers the correct result:

\[ e^+(4,1) \rightarrow p^+(4,1) \]
\[ p^+(4,2) \]
\[ p^+(4,3) \]
\[ p^+(4,4) \]
\[ p^+(3,1) \]
\[ p^+(3,2) \]
\[ p^+(3,3) \]
\[ p^+(3,4) \]
\[ s^+(3) \]
\[ e_{\text{cyclic}}^+ \]
\[ p^+(2,1) \]
\[ p^+(2,2) \]
\[ p^+(2,3) \]
\[ p^+(2,4) \]
\[ s^+(2) \]
\[ e_{\text{cyclic}}^+ \]
\[ p^+(1,1) \]
\[ s^+(1) \]

Now 'p' is needed during FPI, too, due to the effectiveness test!
Dropping Unnecessary Effectiveness Tests

The **s-propagation rule** doesn’t need any effectivity test, as no projection is involved.

The **propagation rule for e_cyclic**, on the other hand, is a projection rule and thus ought to be as follows:

\[ e_{\text{cyclic}}^+ \leftarrow s^+ (X), \text{not } e_{\text{cyclic}}. \]

However, as only consistent states have to be considered (due to integrity checking), the condition 'not e_cyclic' may never be violated.
Negative Propagation Rules: 1st Attempt

\[ R \]

\[
\begin{align*}
e_{\text{cyclic}} & \leftarrow s(X) . \\
s(X) & \leftarrow p(X,X) . \\
p(X,Y) & \leftarrow e(X,Y) . \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y) . \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
p(X,Y) & \leftarrow e(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
\end{align*}
\]

\[ R^\Delta_+ \]

\[
\begin{align*}
e_{\text{cyclic}}^+ & \leftarrow s^+(X) . \\
s^+(X) & \leftarrow p^+(X,X) . \\
p^+(X,Y) & \leftarrow e^+(X,Y), \text{not } p(X,Y) . \\
p^+(X,Y) & \leftarrow e^+(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
p^+(X,Y) & \leftarrow e^+(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
p^+(X,Y) & \leftarrow e^+(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
\end{align*}
\]

Can we transform the original rule set in a similar way as discussed so far for propagating deletions?

\[ U_2: \quad \{ \text{not } e(2,3) \} ! \]

\[ R^\Delta_- \]

\[
\begin{align*}
e_{\text{cyclic}}^- & \leftarrow s^-(X) . \\
s^-(X) & \leftarrow p^-(X,X) . \\
p^-(X,Y) & \leftarrow e^-(X,Y), \text{not } p(X,Y) . \\
p^-(X,Y) & \leftarrow e^-(X,Z), p(Z,Y), \text{not } p(X,Y) . \\
p^-(X,Y) & \leftarrow e^-(X,Z), p^-(Z,Y), \text{not } p(X,Y) . \\
p^-(X,Y) & \leftarrow e^-(X,Z), p^-(Z,Y), \text{not } p(X,Y) . \\
\end{align*}
\]
Negative Propagation Rules: A Bit Different!

- As in consistent states 'e_cyclic' is not derivable, it well never cease to be derivable! Thus, a delta rule for 'e_cyclic?' is not necessary. But then there is no need to compute deletions from 's' either!

- **Combination** of changes is unnecessary, too, as ,e.g., \( e(X,Z) \) always implies \( e(X,Z) \).

- **Effectivity tests** for union resp. projection rules have to be performed **in the new DB-state**! In the old state, all potentially deleted facts were true anyway. In the new state, however, alternative derivations may either have "survived" or have been newly introduced due to insertions taking place simultaneously.
Negative Propagation with Effectiveness Tests

\[ e_\text{cyclic} \leftarrow s(X). \]
\[ s(X) \leftarrow p(X,X). \]
\[ p(X,Y) \leftarrow e(X,Y). \]
\[ p(X,Y) \leftarrow e(X,Z), p(Z,Y). \]

\[ p^{-}(X,Y) \leftarrow e^{-}(X,Y), \text{not } p^{\text{new}}(X,Y). \]
\[ p^{-}(X,Y) \leftarrow e^{-}(X,Z), p(Z,Y), \text{not } p^{\text{new}}(X,Y). \]
\[ p^{-}(X,Y) \leftarrow e(X,Z), p^{-}(Z,Y), \text{not } p^{\text{new}}(X,Y). \]

How to know about the new state while still computing deltas?