Towards a Logical Semantics for Referential Actions (in SQL)

Bertram Ludäscher[I] Wolfgang May[I] Joachim Reinert[II]

[I]Institut für Informatik, Universität Freiburg, Germany

[II]Fachbereich Informatik, Universität Kaiserslautern, Germany

Overview

- Introduction
- Referential Integrity
- Referential Actions in SQL
- Ambiguity: Motivating Examples
- Logical Specification of Referential Actions
- Excursion: Well-Founded Semantics
- Referential Actions as Games
- Statelog: From Specification to Implementation
- Conclusion and Outlook

- *Referential Integrity Constraints* (*RIC*'s) play an important role in relational databases.
- *Referential Actions* (*RAC*'s) are special triggers to automatically maintain referential integrity; now part of the SQL2/3 standards.

Problems (also for more general triggers):

– meaning of *RAC*'s is defined at a very low, procedural level

– rule behavior is difficult to understand

– nondeterministic outcome of *RAC*'s (due to ambiguities) has to be avoided

$\Longrightarrow$ **specify the meaning of RAC's by a logic program P:**

\+ precise and well-defined *high-level* ("declarative") semantics

\+ ambiguities can be avoided

\+ better understanding of rule effects

Referential Integrity Constraints: Example

Emp.(Dept,Proj) references Projects.(Dpt,Prj)

Emp			
E	**Dept**	Sal	**Proj**
$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Projects			
Dpt	$\cdots$	**Prj**	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Notation: A *referential integrity constraint* (*RIC*) is an expression

$$R_C.\overrightarrow{X} \to R_P.\overrightarrow{Y}$$

R_C: **child** relation with foreign key $\overrightarrow{X}$ **references**
R_P: **parent** relation with key $\overrightarrow{Y}$

Here:

R_C = **Emp**, R_P = **Projects**

$\overrightarrow{X}$ = (**Dept, Proj**), $\overrightarrow{Y}$ = (**Dpt, Prj**) *(distinguished arguments)*

$\overline{X}$ = (E, Sal), $\overline{Y}$ = $(\cdots)$ *(remaining arguments)*

Integrity Checking

Every *RIC* $R_C.\overrightarrow{X} \to R_P.\overrightarrow{Y}$ corresponds to a FO-formula φ_{RIC}:

$$\forall \overrightarrow{X}, \overline{X} \,(\, R_C(\overrightarrow{X}, \overline{X}) \to \exists \overrightarrow{Y}, \overline{Y} \,(\overrightarrow{Y} = \overrightarrow{X} \wedge R_P(\overrightarrow{Y}, \overline{Y}))\,) \qquad (\varphi_{RIC})$$

<u>Def.</u> Let D be a relational database ($\approx$ finite FO-structure). Then a *RIC* is **satisfied** if $D \models \varphi_{RIC}$; otherwise if $D \not\models \varphi_{RIC}$, the *RIC* is **violated**.

Integrity Enforcement

If $D \models \varphi_{RIC}$ and $D' := D \pm \Delta$ (Update), then **guarantee** that $D' \models \varphi_{RIC}$.

- simplistic solution: check *(R)IC's* at EOT, abort if violated
- advanced solution: use *active rules* (here: *referential actions/SQL-triggers*) to automatically enforce integrity

Referential Actions in SQL

$$\forall \vec{X}, \overline{X} \, (R_C(\vec{X}, \overline{X}) \rightarrow \exists \vec{Y}, \overline{Y} \, (\vec{Y} = \vec{X} \wedge R_P(\vec{Y}, \overline{Y}))) \tag{φ_{RIC}}$$

Possible Operations: $\left\{ \begin{array}{l} \text{insert into} \\ \text{delete from} \\ \text{update (modify)} \end{array} \right\} \times \left\{ \begin{array}{c} R_P \\ R_C \end{array} \right\}$

not critical:

- insert into R_P (cannot introduce a violation)
- delete from R_C (cannot introduce a violation)
- insert into/update R_C (backed out immediately if violating)

Remaining Operations: $\left\{ \begin{array}{l} \text{delete from} \\ \text{update} \end{array} \right\} \times \left\{ R_P \right\}$

Referential Actions in SQL:

`{CREATE | ALTER} TABLE` R_C

`...`

`FOREIGN KEY` $\vec{X}$ `REFERENCES` R_P $\vec{Y}$

`[ON UPDATE {CASCADE | RESTRICT | SET NULL | SET DEFAULT | NO ACTION}]`

`[ON DELETE {CASCADE | RESTRICT | SET NULL | SET DEFAULT | NO ACTION}]`

`...`

The Problem of Ambiguity: Example

$\boldsymbol{R_A}$

X	$\cdots$
a	$\cdots$
$\cdots$	$\cdots$

$\boldsymbol{R_B}$

X	Y	$\cdots$
a	b	$\cdots$
$\cdots$	$\cdots$	$\cdots$

$\boldsymbol{R_C}$

X	Z	$\cdots$
a	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$

$\boldsymbol{R_D}$

X	Y	Z	$\cdots$
a	b	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

User request $U_{\triangleright} = \{\triangleright\textbf{del:}R_A(a)\} \Longrightarrow$

(1) $R_A \leadsto R_C \leadsto R_D \Longrightarrow$ reject $\triangleright\textbf{del:}R_A(a)$, $D' = D$

(2) $R_A \leadsto R_B \leadsto R_D \Longrightarrow$ accept $\triangleright\textbf{del:}R_A(a)$, $D' = \emptyset$

Computing a Maximal Admissible Subset of $U_{\triangleright}$

R_A

X	$\cdots$
a	$\cdots$
b	$\cdots$
$\cdots$	$\cdots$

R_B

X	Y	$\cdots$
a	b	$\cdots$
b	b	$\cdots$
$\cdots$	$\cdots$	$\cdots$

R_C

X	Z	$\cdots$
a	c	$\cdots$
b	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$

R_E

X	$\cdots$
a	$\cdots$
$\cdots$	$\cdots$

R_D

X	Y	Z	$\cdots$
a	b	c	$\cdots$
b	b	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

User request $U_{\triangleright} = \{\triangleright$**del:**$R_A(a), \triangleright$**del:**$R_A(b)\} \Longrightarrow$
accept $\triangleright$**del:**$R_A(b)$ (similar as before)
reject $\triangleright$**del:**$R_A(a)$ (due to R_E)

Problems with Referential Actions

Nondeterministic outcome of RAC's is undesired.

$\Longrightarrow$ restrict allowed schemas, s.t. ambiguities do not occur, **but:**

Theorem ([Reinert96]) *It is undecidable whether a relational* **schema** *with RIC's and corresponding referential actions (RAC's) contains ambiguities.*

$\Longrightarrow$ (i) find sufficient conditions at the schema level
or (ii) use runtime checks / policy

or **use a more abstract, "declarative" semantics for RAC's by specifying RAC's as logical rules** $\Longrightarrow$

+ logical semantics is precisely defined (no ambiguities due to informal specification)

+ understanding of *RAC*'s

+ if there is no clear "winner", "conflicting" rules lead the third truth-value **undef**; (alternatively: use preferences)

Logical Specification: $\{RAC's\} \mapsto$ logic program P

Relations: $R(\overline{X})$: base relation
$\triangleright$**del:**$R(\overline{X})$: external (user) delete request
req_del:$R(\overline{X})$: internal/induced delete request
blk_del:$R(\overline{X})$: (internal) block request

$\boxed{R_C.\vec{X} \rightarrow R_P.\vec{Y} \ \texttt{ON DELETE RESTRICT}}$ $:\Longleftrightarrow$

$$\textbf{blk\_del:}R_P(\vec{Y},\overline{Y}) \leftarrow R_P(\vec{Y},\overline{Y}),\ R_C(\vec{X},\overline{X}),\ \vec{X} = \vec{Y},\ \neg\,\textbf{req\_del:}R_C(\vec{X},\overline{X}). \tag{DR}$$

$\boxed{R_C.\vec{X} \rightarrow R_P.\vec{Y} \ \texttt{ON DELETE CASCADE}}$ $:\Longleftrightarrow$

$$\textbf{req\_del:}R_C(\vec{X},\overline{X}) \leftarrow \textbf{req\_del:}R_P(\vec{Y},\overline{Y}),\ R_C(\vec{X},\overline{X}),\ \vec{X} = \vec{Y}. \tag{DC_1}$$

$$\textbf{blk\_del:}R_P(\vec{Y},\overline{Y}) \leftarrow \textbf{blk\_del:}R_C(\vec{X},\overline{X}),\ R_P(\vec{Y},\overline{Y}),\ \vec{X} = \vec{Y}. \tag{DC_2}$$

$\boxed{\text{Derive internal requests from user requests}}$ $:\Longleftrightarrow$

$$\textbf{req\_del:}R(\overline{X}) \leftarrow \triangleright\textbf{del:}R(\overline{X}),\ \neg\,\textbf{blk\_del:}R(\overline{X}). \tag{I}$$

Logical Specification of RAC's *(cont'd)*

(+) very concise formalization: $\{RAC's\} \mapsto P$ (logic program)

(+) easy variations, e.g. a more restrictive SQL-like version of (DR):

$$\textbf{blk\_del:}R_P(\overrightarrow{Y},\overline{Y}) \leftarrow R_P(\overrightarrow{Y},\overline{Y}),\ R_C(\overrightarrow{X},\overline{X}),\ \overrightarrow{X} = \overrightarrow{Y},\ \neg\,\textbf{req\_del:}R_C(\overrightarrow{X},\overline{X}).\ (DR^{SQL})$$

(?) what is the "declarative" / "intended" model of P? There are **negative cyclic dependencies**:

$$\textbf{blk\_del:}R \cdots \overset{\neg}{\rightsquigarrow} \cdots \textbf{req\_del:}R' \overset{\neg}{\rightsquigarrow} \cdots \textbf{blk\_del:}R'$$

Excursion: Declarative Semantics for Logic Programs P

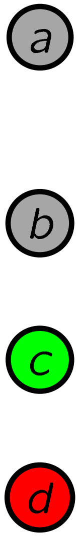
- P positive $\Longrightarrow$ unique minimal model M_P (= least fixpoint of T_P)
- P stratified $\Longrightarrow$ several minimal models, but unique “intended model” (stratified/perfect model) M_P^{supp}
- P non-stratified $\Longrightarrow$
  - **Stable models** ($\geq$ NPTIME):
    Guess model I; reduce negative literals in P wrt $I \rightsquigarrow P/I$.
    I is *stable* :iff $M_{P/I} = I$
    P may have 0, 1, *many* stable models,
  - **Well-founded model** $WFM(P)$ ($\in$ PTIME):
    unique 3-valued ($\{\textbf{true}, \textbf{false}, \textbf{undef}\}$) model,
    every stable model I only extends $WFM(P)$
    $\Longrightarrow WFS$ corresponds to very “sceptical” / “cautious” reasoning

Intuitions of the Well-Founded Semantics (WFS): Games

Consider the program P :

$$win(X) \leftarrow move(X, Y),\ \neg win(Y).$$

P describes a **game** between two players I and II:



move relation

$win(d)$ is	**false**	(d is lost)
$win(c)$ is	**true**	(c is won)
$win(a), win(b)$ are	**undef**	(a, b are drawn)

Playing the "Trigger-Game"

Player I ("Deleter"):

move: $R(\overline{x}) \rightsquigarrow \rhd\textbf{del:}R'(\overline{x}') \quad \Longleftrightarrow$

there is a finite sequence of RAC's `ON DELETE CASCADE` from $\rhd\textbf{del:}R'(\overline{x}')$ to $R(\overline{x})$

claim: *" $R(\overline{x})$ is reachable from $\rhd\textbf{del:}R'(\overline{x}')$ by cascaded deletions "*

Player II ("Blocker"):

move: $\rhd\textbf{del:}R(\overline{x}) \rightsquigarrow R'(\overline{x}') \quad \Longleftrightarrow$

$R'(\overline{x}')$ is blocked by some `ON DELETE RESTRICT` trigger, and there is a sequence of `ON DELETE CASCADE` triggers from $\rhd\textbf{del:}R(\overline{x})$ to $R'(\overline{x}')$

claim: *" $\rhd\textbf{del:}R(\overline{x})$ is referenced by $R(\overline{x})$ which cannot be deleted "*

The Problem of Ambiguity (Revisited)

R_A

X	$\cdots$
a	$\cdots$
$\cdots$	$\cdots$

R_B

X	Y	$\cdots$
a	b	$\cdots$
$\cdots$	$\cdots$	$\cdots$

R_C

X	Z	$\cdots$
a	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$

R_E

X	$\cdots$
a	$\cdots$
$\cdots$	$\cdots$

R_D

X	Y	Z	$\cdots$
a	b	c	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$

I: $R_B(a,b) \leadsto \triangleright$**del:**$R_A(a)$
II: $\triangleright$**del:**$R_A(a) \leadsto R_D(a,b,c)$
I: $R_D(a,b,c) \leadsto \triangleright$**del:**$R_A(a)$
II: $\triangleright$**del:**$R_A(a) \leadsto R_D(a,b,c)$
$\Longrightarrow$ drawn game ($\approx$ ambiguous triggers)

I: $R_B(a,b) \rightsquigarrow \triangleright$**del:**$R_A(a)$
II: $\triangleright$**del:**$R_A(a) \rightsquigarrow R_E(a)$
I: **no move**
$\Longrightarrow$ I loses ($R_B(a,b)$ can't be deleted)

Results: Correctness, Maximality

Theorem

- I can win at $R(\overline{x}) \iff WFS(\textbf{req\_del:}R(\overline{x})) = \textbf{true}$
- II can win at $R(\overline{x}) \iff WFS(\textbf{req\_del:}R(\overline{x})) = \textbf{false}$
- $R(\overline{x})$ is drawn $\iff WFS(\textbf{req\_del:}R(\overline{x})) = \textbf{undef}$
  $\iff WFS(\textbf{blk\_del:}R(\overline{x})) = \textbf{undef}$.

Policies: (A) **restrictive**: delete $R(\overline{X})$ iff $WFS(\textbf{req\_del:}R(\overline{x})) = \textbf{true}$
($\approx$ SQL semantics)

(B) **permissive**: delete $R(\overline{X})$ iff $WFS(\textbf{req\_del:}R(\overline{x})) \in \{\textbf{true}, \textbf{undef}\}$
($\approx$ delete a maximal admissible set of user requests)

Theorem

Correctness: (A),(B) guarantee referential integrity.

Maximality: (B) executes the maximal admissible set of user requests.

Statelog (= **State**s + Data**log**): *From Specification to Implementation*

Production Rules
procedural semantics [AV]
queries, updates

Deductive Rules
declarative (e.g. well-founded) semantics
queries

Statelog Rules
declarative, **state-oriented** semantics
queries, updates

Basic idea: allow access to different database states

$\Longrightarrow$ querying + programming of updates in a unified logical language

$\Longrightarrow$ induced blockings and induced deletions are computed alternately in successive states

$\Longrightarrow$ negative cycles are removed, thus program is stratified

$\Longrightarrow$ standard bottom-up (fixpoint) evaluation

Statelog *(= **States** + **Datalog**): From Specification to Implementation*

Policy (B) (priority to deletions):

$\boxed{R_C.\overrightarrow{X} \rightarrow R_P.\overrightarrow{Y} \texttt{ ON DELETE RESTRICT}}$ $:\Longleftrightarrow$

$$[S+1]\ \textbf{blk\_del:}R_P(\overrightarrow{Y}, \overline{Y}) \leftarrow [S]\ R_P(\overrightarrow{Y}, \overline{Y}),\ R_C(\overrightarrow{X}, \overline{X}),\ \overrightarrow{X} = \overrightarrow{Y},\ \neg\,\textbf{req\_del:}R_C(\overrightarrow{X}, \overline{X}).$$

$\boxed{R_C.\overrightarrow{X} \rightarrow R_P.\overrightarrow{Y} \texttt{ ON DELETE CASCADE}}$ $:\Longleftrightarrow$

$$[S]\ \textbf{req\_del:}R_C(\overrightarrow{X}, \overline{X}) \leftarrow [S]\ \textbf{req\_del:}R_P(\overrightarrow{Y}, \overline{Y}),\ R_C(\overrightarrow{X}, \overline{X}),\ \overrightarrow{X} = \overrightarrow{Y}.$$

$$[S]\ \textbf{blk\_del:}R_P(\overrightarrow{Y}, \overline{Y}) \leftarrow [S]\ \textbf{blk\_del:}R_C(\overrightarrow{X}, \overline{X}),\ R_P(\overrightarrow{Y}, \overline{Y}),\ \overrightarrow{X} = \overrightarrow{Y}.$$

$\boxed{\text{Derive internal requests from user requests}}$ $:\Longleftrightarrow$

$$[S]\ \textbf{req\_del:}R(\overline{X}) \leftarrow [S]\ \triangleright\textbf{del:}R(\overline{X}),\ \neg\,\textbf{blk\_del:}R(\overline{X}).$$

$\Longrightarrow$ 2-valued (stratified) model

$\Longrightarrow$ procedural evaluation can be derived

Conclusion

- Concise and elegant specification of *RAC*'s:
    - WFS (declarative semantics): $\{RAC's\} \mapsto P$
    - $WFS(P) \Leftrightarrow$ "Trigger-Game"
    - $\Leftrightarrow$ Statelog formalization $P_{[S]}$
    - $\rightsquigarrow$ algorithmic description
- Different policies (restrictive, permissive)

Outlook

- Extension to *modify* (`ON UPDATE ...`) *RAC*'s
- Verification of properties of active rules
- Implementation