Foundations of Databases

Relational Query Languages /2

Free University of Bozen – Bolzano, 2004–2005

Thomas Eiter

Institut für Informationssysteme
Arbeitsbereich Wissensbasierte Systeme (184/3)
Technische Universität Wien

http://www.kr.tuwien.ac.at/staff/eiter

(Part of the slides based on material by Leonid Libkin)
Queries with “All”

- Find directors whose movies are playing in all theaters.

\[ \{ \text{dir} \mid \forall (\text{th}, \text{tl}') \in \text{Schedule} \exists \text{tl}, \text{act} \ (\text{Schedule}(\text{th}, \text{tl}) \land \text{Movie}(\text{tl}, \text{dir}, \text{act})) \} \]

- What does it actually mean?

- To understand this, we revisit rule-based queries, and write them in logical notation.
By now, this query is very familiar:

\[
\text{answer}(\text{th}) :\text{– movie}(\text{tl}, \text{’Polanski’}, \text{act}), \text{schedule}(\text{th},\text{tl})
\]

What does it actually mean?

It asks, for each theater (th): “Does there exist a movie (tl) and an actor (act) such that (th,tl) is in Schedule and (tl, ’Polanski’, act) is in Movie?”

This can be stated using notation from mathematical logic:

\[
Q(\text{th}) = \exists \text{tl} \exists \text{act Movie}(\text{tl}, \text{’Polanski’}, \text{act}) \land \text{Schedule}(\text{th},\text{tl})
\]
Other queries in logical notation

- answer(th) :- movie(tl, dir, 'Nicholson'), schedule(th,tl)

- Query as formula:

\[ Q(th) = \exists tl \exists dir \text{ Movie}(tl, dir, 'Nicholson') \land \text{ Schedule}(th,tl) \]

- In general, every single-rule query can be written in the logical notation using only:

  existential quantification \( \exists \), and

  logical conjunction \( \land \) (AND)
SPJRU queries in logical form

- Find actors who played in movies directed by Kubrick OR Polanski.

- Rule-based query:
  
  \[
  \text{answer}(\text{act}) \text{ :- } \text{movie}(tl, \text{dir}, \text{act}), \text{dir='Kubrick'}
  \]
  
  \[
  \text{answer}(\text{act}) \text{ :- } \text{movie}(tl, \text{dir}, \text{act}), \text{dir='Polanski'}
  \]

- Logical notation:

  \[
  Q(\text{act}) = \exists \text{tl} \ \exists \text{dir} \left( \text{Movie}(tl, \text{dir}, \text{act}) \land \left( \text{dir='Kubrick'} \lor \text{dir='Polanski'} \right) \right)
  \]

- New element here: logical disjunction \( \lor \) (OR)

- SPJRU queries can be written in logical notation using: existential quantifiers “\( \exists \)”
  
  conjunction “\( \land \)”
  
  and disjunction “\( \lor \)”
Queries with “for all”

- \{ \text{dir} \mid \forall (\text{th, tl'}) \in \text{Schedule} \exists \text{tl, act} \text{ Schedule}(\text{th,tl}) \land \text{Movie}(\text{tl, dir, act}) \}\}

- New element here: universal quantification “for all” \(\forall\)
- \(\forall x F(x) = \neg \exists x \neg F(x)\)
- So really the new element is: negation
- One has to be careful with negation: what is the meaning of

\[\{x \mid \neg R(x)\}\]

- It seems to say: give us everything that is not in the database. But this is an \textit{infinite} set!
Queries with “all” and negation cont’d

- Safety: a query written in logical notation is safe, it is guaranteed to return finite results on all databases.

- Clearly this has to be enforced in practical languages.

- Bad news: No algorithm exists to check whether a query is safe.

- A bit of good news: All SPJR and SPJRU queries are safe.
  
  Reason: Everything that occurs in the output must have occurred in the input; no new elements are created.

- So we have to figure out how to handle negation.
Relational Calculus

- Relational calculus: queries written in the logical notation using:
  - relation names (e.g., Movie)
  - constants (e.g., 'Nicholson')
  - conjunction $\land$, disjunction $\lor$
  - negation $\neg$
  - existential quantifiers $\exists$
  - universal quantifiers $\forall$

- $\land$, $\exists$, $\neg$ suffice:
  - $\forall x F(x) = \neg \exists x \neg F(x)$
  - $F \lor G = \neg (\neg F \land \neg G)$

- Another name for it: first-order predicate logic.
Relational Calculus cont’d

- Bound occurrence of a variable $x$ in formula $\varphi$: within the scope of a quantifier $\exists x$ or $\forall x$

- free occurrence of a variable in formula $\varphi = \text{not bound occurrence}$

- Free variable of formula $\varphi$: a variable with free occurrence.

- Free variables are those that go into the output of a query.

- Two ways to write a query:
  - $Q(\vec{x}) = F$, where $\vec{x}$ is the tuple of free (distinct) variables
  - $\{\vec{x} \mid F\}$
• Examples:

\[
\{ x, y \mid \exists z \ (R(x, z) \land S(z, y))\}
\]

\[
\{ x \mid \forall y R(x, y)\}
\]

• Queries without free variables are called *Boolean queries*.

• Their output is *true* or *false*

• Examples:

\[
\forall x R(x, x)
\]

\[
\forall x \exists y R(x, y)
\]
Query Semantics

Different ways to define semantics of $Q(\bar{x})$, depending on the range of quantifiers

- **Natural semantics** $Q_{nat}(I)$: unrestricted interpretation, that is, range of quantifiers $\exists x, \forall x$ is $\text{dom}$.

- **Active domain semantics** $Q_{adom}(I)$: range of quantifiers $\exists x, \forall x$ is the set of all constants that occur in the expression $Q$ and in $I$.

- These definitions might lead to different query results.

- Examples:
  \[
  \{x, y, z \mid \neg \text{Movie}(x, y, z)\}
  \]
  \[
  \{x, y \mid \text{Movie}(x, \text{Polanski}, \text{Nicholson}) \lor \text{Movie}(\text{Chinatown}, \text{Polanski}, y)\}
  \]

  The query results are *domain dependent*. 
Query Semantics

- Intuitive Problem: possibly infinite query outputs

- More subtle problem: Range of quantifiers

\[ Q(x) = \{ x \mid \forall y \ R(x, y) \} \]

\[
\begin{array}{l|cc}
R & A & B \\
\hline
a & a \\
a & b \\
\end{array}
\]

- \( Q_{nat}(I) = \emptyset \), while \( Q_{adom}(I) = \{ \langle a \rangle \} \).
**Domain independence**

\[ Q_d(I) \]: Given a query \( Q(\bar{x}) \), a set \( d \subseteq \text{dom} \), and a database instance \( I \) such that all constants in \( Q \) and in \( I \) occur in \( d \). Then \( Q_d(I) \) denotes the evaluation of \( Q(\bar{x}) \) on \( I \) (aka image of \( I \) under \( Q(\bar{x}) \)) relative to \( d \), i.e., free variable and quantifiers range over \( d \).

**Defn.** A query \( Q(\bar{x}) \) is *domain independent*, if for all \( d, d' \) and \( I \), \( Q_d(I) = Q_{d'}(I) \) (whenever both are defined).

- **Positive examples:**
  \[ \exists \ t l \ \exists \ \text{act} \ \text{Movie}(tl, 'Polanski', \text{act}) \land \text{Schedule}(th,tl) \]
  Every SPJU query (rewritten to logical notation)

- **Negative examples:**
  \[ \{x, y, z \mid \neg \text{Movie}(x, y, z)\} \]
  \[ \{x, y \mid \text{Movie}(x,\text{Polanski},\text{Nicholson}) \lor \text{Movie}(\text{Chinatown},\text{Polanski},y)\} \]
**Proposition.** If $Q(\vec{x})$ is domain independent, then for each $d \subseteq \text{dom}$ and database instance $I$ such that $Q_d(I)$ is defined,

$$Q_d(I) = Q_{\text{nat}}(I) = Q_{\text{adom}}(I)$$

**Defn.** Domain-independent Relational Calculus (DI-RelCalc) = set of domain-independent queries in RC.

- Drawback: domain independence is not a recursive notion.
- That is, it is undecidable whether a given formula $Q(\vec{x})$ belongs to DI-RelCalc.
- Still, there is syntax for domain-independent queries
- Syntactic fragments of DI-RelCalc which are as expressive as RelCalc, like safe *range queries*, can be efficiently recognized.
Relational Algebra: Difference

- If $R$ and $S$ are two relations with the same set of attributes, then $R - S$ is their difference:

  The set of all tuples that occur in $R$ but not in $S$.

- Example:

  $R = \begin{array}{cc} A & B \\ a1 & b1 \\ a2 & b2 \\ a3 & b3 \end{array}$ and $S = \begin{array}{cc} A & B \\ a2 & b2 \\ a3 & b3 \\ a4 & b4 \end{array}$

  $R - S = \begin{array}{cc} A & B \\ a1 & b1 \end{array}$
**Fundamental Theorem of Relational Database Theory**

**Theorem.**

Domain-independent Relational Calculus (DI-RelCalc)

\[ \text{\begin{align*}
\text{\; Domain-independent Relational Calculus (DI-RelCalc) } & = \text{ Relational Calculus under Active Domain Semantics } \\
& = \text{ Relational Algebra with operations } \pi, \sigma, \times, \cup, \setminus, \rho
\end{align*}} \]

- We won’t give a formal proof of this statement, but try to explain why it is true.
  
  Side effect: see some examples of relational algebra programming
Show that relational algebra can be expressed by relational calculus

- Use only $\exists$ quantifier in mapping

- Each free variable $x$ and resp. quantified variable $\exists x$ must be “grounded” in some atom $R(..., x, ...)$

- Thus, for each RA expression $e$ the semantics of its transform $F_e$ is wolog. the Active Domain Semantics.
From Relational Algebra to DI-RelCalc/2

- Each expression $e$ producing an $n$-attribute relation is translated into a formula
  \[ F_e(x_1, \ldots, x_n) \]
- $R \rightarrow R(x_1, \ldots, x_n)$
- $\sigma_c(R) \rightarrow R(x_1, \ldots, x_n) \wedge c$

Example: if $R$ has attributes $A, B$ then $\sigma_{A=B}(R)$ is translated into
  \[ (R(x_1, x_2) \wedge x_1 = x_2) \].
From Relational Algebra to DI-RelCalc/3

• If $R$ has attributes $A_1, \ldots, A_n, B_1, \ldots, B_m$, then
  
  $$\pi_{A_1, \ldots, A_n}(R)$$

  is translated into

  $$\exists y_1, \ldots, y_m \ R(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

  Important: it is the attributes that are not projected that are quantified.

  Example: for $R$ with attributes $A, B$, $\pi_A(R)$ is $\exists x_2 R(x_1, x_2)$.

• $R \times S$ is translated into

  $$R(x_1, \ldots, x_n) \land S(y_1, \ldots, y_m)$$

  (note that all the variables are distinct; hence the output will have $n + m$ attributes)
From Relational Algebra to DI-RelCalc/4

- If $R$ and $S$ both have the same attributes, then $R \cup S$ is translated into

$$R(x_1, \ldots, x_n) \lor S(x_1, \ldots, x_n)$$

(note that all the variables are the same, hence the output will have $n$ attributes)

- If $R$ and $S$ both have the same attributes, then $R - S$ is translated into

$$R(x_1, \ldots, x_n) \land \neg S(x_1, \ldots, x_n)$$

(note that all the variables are the same, hence the output again will have $n$ attributes)
Getting ready for DI-RelCalc to algebra translation

- **Active domain** of a relation: the set of all constants that occur in it.

  \[
  R_1 \begin{array}{cc}
  A & B \\
  a_1 & b_1 \\
  a_2 & b_2 \\
  \end{array}
  \]

  has active domain \(\{a_1, a_2, b_1, b_2\}\).

- We can compute the active domain of \(R\) in RA:

  Suppose \(R\) has attributes \(A_1, \ldots, A_n\).

  \[
  ADOM(R) = \rho_B \leftarrow A_1 \left( \pi_{A_1}(R) \right) \cup \ldots \cup \rho_B \leftarrow A_n \left( \pi_{A_n}(R) \right)
  \]

  - It is a relation with one attribute \(B\).

  - Similarly we can compute

  \[
  ADOM(R_1, \ldots, R_k) = ADOM(R_1) \cup \ldots \cup ADOM(R_k)
  \]
A domain-independent query $Q(\bar{x})$ over relations $R_1, \ldots, R_n$ can be wlog. be evaluated over $\text{ADOM}(R_1, \ldots, R_n)$

We thus translate relational calculus queries evaluated within $\text{ADOM}(R_1, \ldots, R_n)$ into relational algebra queries.

Each relational calculus formula $F(x_1, \ldots, x_n)$ is translated into an expression $E_F$ that produces a relation with $n$ attributes.
From DI-RelCalc to relational algebra /2

- Easy cases (for $R$ with attributes $A_1, \ldots, A_n$):
  
  $$R(x_1, \ldots, x_n) \rightarrow R$$
  
  $$\exists x_1 R(x_1, \ldots, x_n) \rightarrow \pi_{A_2, \ldots, A_n}(R)$$

- Not so easy cases:

- condition $c(x_1, \ldots, x_n)$ is translated into

  $$\sigma_c(\text{ADOM} \times \ldots \times \text{ADOM})$$

  E.g., $x_1 = x_2$ is translated into $\sigma_{x_1=x_2}(\text{ADOM} \times \text{ADOM})$

- Negation $\neg R(\bar{x}) \rightarrow \text{ADOM} \times \ldots \times \text{ADOM} - R$

  That is, we only compute the tuples of elements from the database that do not belong to $R$
The hardest case: disjunction

Let both $R$ and $S$ have two attributes.

Relational calculus query: $Q(x, y, z) = R(x, y) \lor S(x, z)$

Its result has three attributes, and consists of tuples $(x, y, z)$ such that

either $(x, y) \in R, z \in \text{ADOM},$ or $(x, z) \in S, y \in \text{ADOM}$

The first one is simply $R \times \text{ADOM}$

The second one is more complex:

$\pi_{#1,#3,#5}(\sigma_{#1=#4\land#2=#5}(S \times \text{ADOM} \times S))$

Thus, $Q$ is translated into

$R \times \text{ADOM} \cup \pi_{#1,#3,#5}(\sigma_{#1=#4\land#2=#5}(S \times \text{ADOM} \times S))$
Alternative: Mapping conjunction using natural join

Suppose we have relations $R : A_1, \ldots, A_m, B_1, \ldots, B_n$ and $S : A_1, \ldots, A_m, C_1, \ldots, C_k$ for formulas $\varphi(x_1, \ldots, x_m, y_1, \ldots, y_n)$ and $\psi(x_1, \ldots, x_m, z_1, \ldots, z_k)$, respectively.

Then $\varphi(x_1, \ldots, x_m, y_1, \ldots, y_n) \land \psi(x_1, \ldots, x_m, z_1, \ldots, z_k)$ is mapped to

$$R \bowtie S$$

The natural join can be defined in terms of $\times$, $\sigma$, and $\rho$. 
Queries with “all” in relational algebra revisited

- Find directors whose movies are playing in all theaters.

\[
\{ \text{dir} \mid \forall (\text{th}, \text{tl}') \in \text{Schedule} \exists \text{tl}, \text{act} \ \text{Schedule}(\text{th},\text{tl}) \land \text{Movie}(\text{tl}, \text{dir}, \text{act}) \}\n\]

- Define:

\[
T_1 = \pi_{\text{theater}}(S) \quad T_2 = \pi_{\text{theater}, \text{director}}(M \bowtie S)
\]

(to save space, we use \(M\) for Movie and \(S\) for Schedule)

- \(T_1\) has all theaters, \(T_2\) has all directors and theaters where their movies are playing.

- Our query is:

\[
\{d \mid \forall t \in T_1 \ (t, d) \in T_2\}\]
Queries with “all” cont’d

Query \{d \mid \forall t \in T_1 \land T_2(t, d)\} is rewritten to

\{d \mid \neg (\exists t \in T_1 (t, d) \notin T_2)\}

Hence, the answer to the query is

\pi_{\text{director}}(M) - V

where \( V = \{d \mid (\exists t \in T_1 (t, d) \notin T_2)\} = \{d \mid \exists t T_1(t) \land \neg T_2(t, d)\} \).

Pairs (theater, director) not in \( T_2 \) are

\( T_1 \times \pi_{\text{director}}(M) - T_2 \)

Thus

\begin{align*}
V &= \pi_{\text{director}}(T_1 \times \pi_{\text{director}}(M) - T_2)
\end{align*}
Reminder: the query is
Find directors whose movies are playing in all theaters.

Putting everything together, the answer is:

$$\pi_{\text{director}}(M) - \pi_{\text{director}}\left(\pi_{\text{theater}}(S) \times \pi_{\text{director}}(M) - \pi_{\text{theater,director}}(M \Join S)\right)$$

This is much less intuitive than the logical description of the query.

Indeed, procedural languages are not nearly as comprehensible as declarative.
Safe-Range Queries

- A syntactic fragment of Relational Calculus which contains only domain-independent queries (and thus also a fragment of DI-RelCalc)

- Safe-Range RelCalc = DI-RelCalc

- Involves
  1. a syntactic normal form of the queries
  2. a mechanism for determining whether a variable is range restricted
  3. a global property to be satisfied
Safe-Range Normal Form (SRNF)

Rewrite query formula $Q(\bar{x})$ without substantially changing its structure

- Variable substitution: Replace variables such that each variable $x$ is quantified at most once and has only free or only bound occurrences.
- Remove $\forall$: Rewrite $\forall \varphi$ to $\neg \exists \neg \varphi$
- Remove implications: Rewrite $\varphi \Rightarrow \psi$ to $\neg \varphi \lor \psi$, and similarly for $\iff$
- Push negation inside as much as possible, using
  \[
  \neg \neg \varphi \rightarrow \varphi
  \]
  \[
  \neg (\varphi_1 \land \varphi_2) \rightarrow \neg \varphi_1 \lor \neg \varphi_2
  \]
  \[
  \neg (\varphi_1 \lor \varphi_2) \rightarrow \neg \varphi_1 \land \neg \varphi_2
  \]
- Flatten ‘and’s: No child of an ‘and’ in the formula parse tree is an ‘and’. Similarly for ‘or’s, and ‘$\exists$’s (this step is not essential)
Resulting formula: $SRNF(Q(\bar{x}))$

Query $Q(\bar{x})$ is in safe-range normal form if $SRNF(Q(\bar{x})) = Q(\bar{x})$

Examples:

$$Q_1(\text{th}) = \exists \text{tl} \exists \text{dir Movie}(\text{tl, dir,'Nicholson'}) \land \text{Schedule}(\text{th,tl})$$

$$SRNF(Q_1) = \exists \text{tl, dir Movie}(\text{tl, dir,'Nicholson'}) \land \text{Schedule}(\text{th,tl})$$

$$Q_2(\text{dir}) = \forall \text{th} \forall \text{tl'} (\text{Schedule}(\text{th,tl'}) \rightarrow (\exists \text{tl} \exists \text{act Schedule}(\text{th,tl}) \land \text{Movie}(\text{tl, dir, act})))$$

$$SRNF(Q_2) = \neg\exists \text{th, tl'} \text{Schedule}(\text{th,tl'}) \lor (\exists \text{tl, act Schedule}(\text{th,tl}) \land \text{Movie}(\text{tl, dir, act}))$$
Range Restriction

- Syntactic condition on formulas in SRNF.

- Intuition: all possible values of a variable lie in the active domain.

- If a variable doesn’t fulfill this, then the query is rejected
Algorithm Range Restriction (rr)

Input: formula $\varphi$ in SRNF
Output: subset of the free variables or $\bot$

**case** $\varphi$ of

- $R(e_1, \ldots, e_n)$: $rr(\varphi) := \text{the set of variables from } e_1, \ldots, e_n$.
- $x = a, a = x$: $rr(\varphi) := \{x\}$
- $\varphi_1 \land \varphi_2$: $rr(\varphi) := rr(\varphi_1) \cup rr(\varphi_2)$
- $\varphi_1 \land x = y$: $\text{if } \{x, y\} \cap rr(\varphi_1) = \emptyset$
  - then $rr(\varphi) := rr(\varphi_1)$ else $rr(\varphi) := rr(\varphi_1) \cup \{x, y\}$
- $\varphi_1 \lor \varphi_2$: $rr(\varphi) := rr(\varphi_1) \cap rr(\varphi_2)$
- $\neg \varphi_1$: $rr(\varphi) := \emptyset$
- $\exists x_1, \ldots, x_n \varphi_1$: $\text{if } \{x_1, \ldots, x_n\} \subseteq rr(\varphi_1)$ then $rr(\varphi) := rr(\varphi_1) \setminus \{x_1, \ldots, x_n\}$ else return $\bot$

**end case**

Here, $S \cup \bot = \bot \cup S = \bot$ and similarly for $\cap, \setminus$
Example (cont’d):

\[
\begin{align*}
SRNF(Q_1) &= \exists tl, \text{dir } \text{Movie}(tl, \text{dir},'Nicholson') \land \text{Schedule}(th,tl) \\
rr(SRNF(Q_1)) &= \{th\} \\
SRNF(Q_2) &= \neg\exists th, tl' \text{ Schedule}(th,tl') \lor (\exists tl, \text{act } \text{Schedule}(th,tl) \land \text{Movie}(tl, \text{dir}, \text{act})) \\
rr(SRNF(Q_2)) &= \{\}
\end{align*}
\]

**Defn.** A query \(Q(\bar{x})\) in Relational Calculus is *safe-range* if \(rr(SRNF(Q))\) coincides with the set of free variables in \(Q\). The set of all safe-range queries is denoted by SR-RelCalc.

Examples: \(Q_1\) is a safe-range query, while \(Q_2\) is not.

**Theorem.** SR-RelCalc = DI-RelCalc
For all and negation in SQL

- Two main mechanisms: subqueries, and Boolean expressions
- Subqueries are often more natural
- SQL syntax for $R \cap S$:
  
  ```sql
  R INTERSECT S
  ```
- SQL syntax for $R - S$:
  
  ```sql
  R EXCEPT S
  ```
- Find all actors who are not directors resp. also directors:
  
  ```sql
  SELECT Actor AS Person FROM Movie EXCEPT SELECT Director AS Person FROM Movie INTERSECT
  ```

```sql
SELECT Actor AS Person FROM Movie EXCEPT SELECT Director AS Person FROM Movie INTERSECT
```
For all and negation in SQL/2

Subqueries with \texttt{NOT EXISTS}, \texttt{NOT IN}

- Example: Find directors whose movies are playing in all theaters.

- SQL’s way of saying this: Find directors such that there does not exist a theater where their movies do not play.

```sql
SELECT M1.Director
FROM Movie M1
WHERE NOT EXISTS (SELECT S.Theater
                   FROM Schedule S
                   WHERE NOT EXISTS (SELECT M2.Director
                                       FROM Movie M2
                                       WHERE M2.Title=S.Title AND
                                             M1.Director=M2.Director))
```
Bibliography


