Relational Algebra

Elisa Bertino
CS Department and CERIAS
Purdue University

Operations for the Relational Model

- Operations on relations can be expressed according to two basic formalisms:
  - Relational Algebra: queries are specified by applying specialized operators to relations
  - Relational Calculus: queries are expressed through logical formulas that must be verified by the tuples that are result of queries
  - The two formalisms (under certain assumptions) are equivalent

Relational Algebra

- It consists of five basic operations:
  - Union
  - Difference
  - Cartesian Product
  - Projection
  - Selection
- These operations fully define the relational algebra

Relational Algebra

- Each operation returns a relation as result; it is thus possible to apply an operation to the result of another operation (closure property)
- There are some additional operations that can be expressed in terms of the five basic operations
  - Join
  - Intersection
  - Division
Relational Algebra

- Such additional operations do not increase the **expressive power** of the set of the basic operations
- However, these additional operations are used as shorthand; the **join** is the most important of these operations
- With respect to the notation by name, it can be useful to introduce another operation, called **renaming**, that allows one to modify the attribute names

Relational Algebra - Union

- The union of relations R and S, denoted by $R \cup S$
  - is the set of tuples that are in R, in S or in both
- The union of two relations can only be executed if the relations have the *same degree*; in addition the first attribute of R must be *compatible* with the first attribute of S, the second attribute of R must be *compatible* with the second attribute of S, and so forth
- The duplicate tuples are eliminated
- The degree of the resulting relation is the same as the degree of the input relations

**Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

B | G | A

R ∪ S

Relational Algebra - Difference

- The difference of relations R and S, denoted as $R - S$
  - is the set of tuples that are in R and are not in S
- The difference, like the union, can only be executed if the input relations have the same degrees and compatible attributes
- The degree of the resulting relation is the same as the degree of the input relations
Relational Algebra – Difference Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
</tbody>
</table>

R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

R-S

Relational Algebra - Cartesian Product

- The Cartesian product of relations R and S, having degree \( k_1 \) and \( k_2 \), denoted as \( R \times S \), is a relation of degree \( k_1 + k_2 \); the tuples of such relation are all the tuples having:
  - as first \( k_1 \) components a tuple of R
  - as second \( k_2 \) components a tuple of S

In the resulting relation the names of the first \( k_1 \) attributes are the names of the attributes of R and the names of the last \( k_2 \) attributes are the names of the attributes of S.

If the two relations have some attributes with the same name, the names of such attributes in one of the two relations must be renamed.

Relational Algebra - Cartesian Product Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
</tbody>
</table>

R

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
</tbody>
</table>

S

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
<td>b</td>
<td>g</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

R x S
Relational Algebra - Projection

- Let $R$ be a relation and $A=\{A_1,\ldots,A_m\} \subseteq U_R$ be a set of attributes of $R$. The projection of $R$ on $A$, denoted as
  \[ \Pi_{A_1,\ldots,A_m}(R) \]
  is a relation of degree $m$; the tuples of such relation have as attributes only the attributes in $A$.
- The projection generates a set $T$ of $m$-tuples such that if $t=[A_1:v_1,\ldots,A_m:v_m] \in T$, then a tuple $t'$ exists in $R$ such that for each $A_i \in A$, $1 \leq i \leq m$, $t[A_i]=t'[A_i]$.
- In the resulting relation, the attributes have the order specified in $A$.

Relational Algebra – Projection

Example

\[
\begin{array}{ccc}
A & B & C \\
--- & --- & --- \\
a & b & c \\
d & a & f \\
c & b & d \\
\end{array}
\]

\[
\begin{array}{cc}
A & C \\
b & c \\
d & f \\
c & d \\
\end{array}
\]

\[
\begin{array}{cc}
B & A \\
b & a \\
a & d \\
b & c \\
\end{array}
\]

Relational Algebra - Predicates

A predicate $F$ on a relation $R$ has one of the following forms:
- Simple predicate – it has one of the following forms
  \[ A \; op \; k \; (1) \]
  \[ A \; op \; A' \; (2) \]
  where $A$ and $A'$ are attributes of $R$, $op$ is a relational comparison operator ($>,<,\leq,\geq,=,\text{etc.}$), $k$ is a constant value compatible with the domain of $A$.
- Boolean combination of simple predicates; such combinations are specified through the use of the logical operators \(\land\), \(\lor\) and \(\neg\).

Relational Algebra – Predicates

Examples

- $B=b$ simple predicate of form (1)
- $A=C$ simple predicate of form (2)
- $B=b \lor A=C$ Boolean combination
- $B=b \land A=C$ Boolean combination
- $\neg B=b$ Boolean combination
Relational Algebra - Selection

- The selection on a relation $R$ given a predicate $F$ defined on $R$, denoted as
  \[ \sigma_F(R) \]
  is a relation containing all and only the tuples of $R$ that verify predicate $F$.
- The degree of the resulting relation is the same of the degree of the input relation.
- If no tuple of $R$ verifies $F$, the result is the empty relation.

Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

$R$

σ₈₆₇(R)

$σ_{\neg B=b}(R)$

$σ_{A=\neg B=b}(R)$

$σ_{A=C}(R)$

Relational Algebra - Renaming

- The renaming of a relation $R$ with respect to a list of pairs of attribute names $(A_1, B_1), \ldots, (A_m, B_m)$ such that $A_i, 1 \leq i \leq m$, is a name of an attribute of $R$, denoted with
  \[ \rho_{A_1, \ldots, A_m \leftarrow B_1, \ldots, B_m}(R) \]
  replace the name of attribute $A_i$ with the name $B_i$.
- The renaming is correct if the new schema of relation $R$ has all attributes with distinct names.
Relational Algebra – Renaming

Example

- The renaming:

$$\rho_{A,B,C \leftarrow AA,BB,CC}(R)$$
changes the schema $$R(A,B,C)$$ into $$R(AA,BB,CC)$$.

Employees

<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>Job</th>
<th>HiringD</th>
<th>Salary</th>
<th>Bonus</th>
<th>Dept#</th>
</tr>
</thead>
<tbody>
<tr>
<td>7584</td>
<td>Pink</td>
<td>Manager</td>
<td>02/04/81</td>
<td>2975.00</td>
<td>?</td>
<td>20</td>
</tr>
<tr>
<td>7698</td>
<td>Martin</td>
<td>Secretary</td>
<td>20/02/81</td>
<td>800.00</td>
<td>100.00</td>
<td>30</td>
</tr>
<tr>
<td>7756</td>
<td>White</td>
<td>Technician</td>
<td>01/06/81</td>
<td>800.00</td>
<td>?</td>
<td>30</td>
</tr>
<tr>
<td>7839</td>
<td>Dare</td>
<td>Engineer</td>
<td>17/11/81</td>
<td>2600.00</td>
<td>300.00</td>
<td>10</td>
</tr>
<tr>
<td>7844</td>
<td>Green</td>
<td>Manager</td>
<td>10/12/80</td>
<td>3000.00</td>
<td>?</td>
<td>10</td>
</tr>
</tbody>
</table>

Relational Algebra – Example

- Q1: determine the name of the employees having a salary greater than 2000

$$\Pi_{Name}(\sigma_{Salary>2000}(Employee))$$

Name
Pink
Black
Neri
Dare
Green

- Q2: determine the name and the department number of the employees that are engineers and have a salary greater than 2000

$$\Pi_{Name,Dept\#}(\sigma_{Salary>2000 \land Job='Engineer'}(Employees))$$

Name
 Dept#
Neri 10
Dare 10
Relational Algebra – Join

Let R and S be relations; let A be an attribute of R and A’ an attribute of S such that the domains of the attributes are compatible; let θ be a relational comparison operator. The join of R and S with respect to the predicate AθA’ is denoted as R \Join_{A \theta A’} S and is defined as σ_{A \theta A’}(R \times S).

The join is thus a Cartesian product followed by a selection.

The predicate AθA’ is called join predicate.

The degree of the resulting relation is equal to the sum of the degrees of the input relations.

Alternative notations for the join:
- R.AθS.A’
- R{AθA’}S

The join is called equijoin when the θ operator is the equality operator.

Examples

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

R \times S

R \Join_{A \theta A’} S

Relational Algebra – Natural Join

It is a simplification of the join; it is very frequent.

It is a join based on the equality of the values for attributes that are in common among the input relations.

Unlike the other RA operations, it is meaningful only when the notation by name is used.
Relational Algebra – Natural Join

- Let:
  - \( R \) and \( S \) be relations
  - \( \{A_1, \ldots, A_k\} = U R \cap US \) be the set of attribute names present in both the schema of \( R \) and the schema of \( S \)
  - \( \{B_1, \ldots, B_m\} = U R \cup US \) be the set of attribute names present in the schema of \( R \) or in the schema of \( S \)
  - The expression that defines the natural join is
    \[ \Pi_{B_1, \ldots, B_m}(\sigma_C(R \times (\rho_{A_1, \ldots, A_k \leftarrow S.A_1, \ldots, S.A_k}(S)))) \]
    where \( C \) is a predicate of the form:
    \( A_1 = S.A_1 \land \ldots \land A_k = S.A_k \)

- The natural join thus performs a join by equating the attributes with the same name and then eliminates the replicated attributes
- The natural join is denoted as \( R \times S \)

**Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

\( R \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>f</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
</tr>
</tbody>
</table>

\( S \)

**Relational Algebra – Intersection**

- The intersection of two relation \( R \) and \( S \), denoted as \( R \cap S \)
  - is defined as
    \[ R \cap S = R \setminus (R \setminus S) \]
  - The same restrictions of the difference apply to the input relations
Suppose that employees of the company are assigned to some professional courses; each employee follows several courses and each course is taken by several employees.

Information about courses and which employee takes which course can be represented by the following relations:

- Takes (Emp#, Course#)
- Courses (Course#, Topic, Weeks)

Consider the following query: "determine the emp# of the employees taking all the courses with topic DB" (Q1)

The course# of the courses relevant to the query is obtained through the following expression:

\[ R_1 = \Pi_{\text{Course#}}(\sigma_{\text{Topic} = \text{DB}}(\text{Courses})) \]

The result of such query is \{10, 30\}

The result of query Q1 is represented by all the employees that appear in the Takes relation with each of the course numbers retrieved by query R1.

The correct result of Q1 is thus \{7369\}

The operation that allows one to execute such query is the division.
Relational Algebra – Division

Let $R$ and $S$ be relations; let $U_R$ and $U_S$ be the attribute sets of $R$ and $S$, respectively, and such that $U_R \supset U_S$.

The division operation is denoted as $R \div S$ and is expressed as follows:

$$
\Pi_{U_R - U_S}(R) - \Pi_{U_R - U_S}((\Pi_{U_R - U_S}(R) \times S) – R)
$$

The expression on the right side of the $-$ sign determines all tuples of $R$ that are not associated with at least a tuple of $S$.

Example

Query Q1 is thus expressed as follows:

$$
\text{Takes} \div \Pi_{\text{Course#}}(\sigma_{\text{Topic} = \text{DB}}(\text{Courses}))
$$

$R = \text{Takes}$

$S = \Pi_{\text{Course#}}(\sigma_{\text{Topic} = \text{DB}}(\text{Courses}))$

$S = \{10, 30\}$

$U_R = \{\text{Emp#}, \text{Course#}\}$

$U_S = \{\text{Course#}\}$

Example

1) $\Pi_{U_R - U_S}(R) = \Pi_{\text{Emp#}}(R) = \{7369, 7782\}$

2) $\Pi_{U_R - U_S}(R) \times S = \Pi_{\text{Emp#}}(R) \times S = \{(7369, 10), (7369, 30), (7782, 10), (7782, 30)\}$

3) $(\Pi_{U_R - U_S}(R) \times S) - R = (\Pi_{\text{Emp#}}(R) \times S) - R = \{(7369, 10), (7369, 30), (7782, 10), (7782, 30)\}$

4) $\Pi_{U_R - U_S}((\Pi_{U_R - U_S}(R) \times S) - R) = \Pi_{\text{Emp#}}((\Pi_{\text{Emp#}}(R) \times S) - R) = \{7782\}$
Relational Algebra – Division

Example

5) \( \Pi_{(U \cup U \setminus S)}(R) \) - computed at step 1
\( \Pi_{(U \cup U \setminus S)}((\Pi_{(U \cup U \setminus S)}(R) \times S) - R) \) - computed at step 4

\[
\begin{array}{ccc}
\text{Emp#} & \text{Emp#} & \text{Emp#} \\
7369 & 7782 & = 7369 \\
7782 & & \\
\end{array}
\]

Relational Calculus

Elisa Bertino
CS Department and CERIAS
Purdue University

Algebra vs. Calculus

- The relational algebra is a "procedural" language: when specifying an algebraic expression we need to specify the operations to be used to compute the query result and the order according to which these operations have to be executed.
- The relational calculus is declarative; we just need to give a formal definition of the query result, without specifying how to obtain it.

Calculus – variations

- There are two variations
  - Tuple relational calculus (TRC)
    - The variables denote tuples
    - Note: in SQL variables can be used; such variables denote tuples
  - Domain relational calculus (DRC)
    - The variables denote domain values
    - We will see the TRC only
In TRC a query is an expression of the form
\( \{ t : U | P(t) \} \)
that is, a query is defined as the set of all tuples defined on a set \( U \) of attributes such that predicate \( P \) is true for \( t \).

**Notation:**
- \( t.A \) denotes the value of tuple \( t \) for the attribute \( A \)
- \( t \in R \) denotes that \( t \) belongs to relation \( R \)

**Example:** Retrieve all employees having a salary greater than 2000
\( \{ t : \text{Employees} | t.Salary > 2000 \} \)

- In the example \( t \) is a variable that denotes tuples belonging to a relation that has as schema \( \{ \text{Name} \} \)
- The notation \( \exists s(Q(s)) \) denotes that there exist a tuple \( t \) such that \( Q(s) \) is true

**Examples**
- Retrieve the name and the office of all employees having a salary greater than 2000
\( \{ \{ \text{Name}, \text{Office#} \} | \exists s(s \in \text{Employees} \land s.Salary > 2000 \land t.\text{Name} = s.\text{Name} \land \exists u(u \in \text{Departments} \land s.\text{Dept#} = u.\text{Dept#} \land t.\text{Office#} = u.\text{Office#}) \} \)
- Retrieve the name of all employees having a salary greater than 2000 or working for a department in division D1
\( \{ \{ \text{Name} \} | \exists s(s \in \text{Employees} \land t.\text{Name} = s.\text{Name} \land (s.\text{Salary} > 2000 \lor \exists u(u \in \text{Departments} \land s.\text{Dept#} = u.\text{Dept#} \land u.\text{Division} = 'D1')) \} \)

**TRC – Syntax Atoms**

- The atoms are as follows
  - \( s \in R \), where \( R \) is a relation name and \( s \) is a variable
  - The tuple \( s \) belongs to relation \( R \)
  - \( s.A \theta u.A' \), where \( s \) and \( u \) are variables, \( \theta \) is a relational comparison operator, \( A \) and \( A' \) are attribute names
  - The value of attribute \( A \) in tuple \( s \) is in relation \( \theta \) with the value of attribute \( A' \) in tuple \( u \)
  - \( s.\theta a \), where \( s \) is a variable, \( \theta \) is a relational comparison operator, \( a \) is a constant value
  - The value of attribute \( A \) in tuple \( s \) is in relation \( \theta \) with the constant value \( a \)
Each atom is a formula
all variable occurrences in the atom are free
Let \( \Phi \) be a formula, then (\( \Phi \)) is a formula
Let \( \Phi \) be a formula, then:
\( \exists s(\Phi) \) is a formula; \( \forall s(\Phi) \) is a formula
all occurrences of \( s \) in \( \Phi \) are bound to the quantifier
Let \( \Phi \) and \( \Psi \) be formulae, then:
\( \Phi \land \Psi \) is a formula; \( \Phi \lor \Psi \) is a formula; \( \neg \Phi \) is a formula
The occurrences of variables are free or bound depending on whether they are free or bound in \( \Phi \) and \( \Psi \)

Let \( F \) be a formula and \( x \) be a variable
\( x \) is said to be free in \( F \) if \( x \) is not quantified
\( \exists x \) – existential quantifier
\( \forall x \) – universal quantifier
Example:
\( \exists (s \in \text{Employees} \land s.\text{Salary} > 2000 \land x.\text{Dept#} = y.\text{Dept#}) \)
is a formula which correct according to the syntax
all occurrences of variable \( s \) are bound
all occurrences of variables \( x \) and \( y \) are free

An expression (or query) of the TRC has the form
\( \{x:U \mid F(x)\} \)
where:
- \( U \) is a set of attributes
- \( F \) is a formula, which is correct according to the syntax
- \( x \) is a free variable in \( F \)
- \( x \) is the only free variable in \( F \)

The expression
\( \{y:U_{\text{Employees}} \mid \forall y(y \in \text{Employees} \land y.\text{Job} = \text{Engineer})\} \)
is not a correct expression in that \( y \) is not a free variable
Expressing the Relational Algebra in TRC

- Union: \( R \cup S \) \( \{ t : U_R \ | \ t \in R \lor t \in S \} \)
- Difference: \( R - S \) \( \{ t : U_R \ | \ t \in R \land \neg t \in S \} \)
- Cartesian Product:
  - \( R \times S \) let \( U_R = \{ A_1, \ldots, A_n \} \) and \( U_S = \{ A'_1, \ldots, A_m \} \)
  - be the sets of attributes of \( R \) and \( S \)
  - \( \{ t : U_R \cup U_S \ | \ \exists x \exists y (x \in R \land y \in S \land x. A_1 = t. A_1 \land \ldots \land x. A_n = t. A_n \land y. A'_1 = t. A'_1 \land \ldots \land y. A'_m = t. A'_m) \} \)

TRC and Relational Algebra Expressive power

- Do the relational algebra and the TRC have the same expressive power?
- That is: can all operations that can expressed in the relational algebra also be expressed in TRC and vice-versa?

TRC and Relational Algebra Expressive power

- The semantics of a query (expressed in relational algebra or TRC) is a function that transforms a relational database (a set of relations) into another relational database
- The relational algebra and TRC have the same expressive power if for each query \( Q_1 \) expressed in one of these two formalisms, a query \( Q_2 \) exists expressed in the other formalism, such that the two queries have the same semantics
Not all the expressions of TRC can be expressed in equivalent expressions of the relational algebra.

Consider the expression
\[ \{ t:U_R \mid \neg t \in R \} \]
Even though this expression is syntactically correct, if one of the domains of the attributes of \( R \) is an infinite set, the expression is satisfied by an infinite number of tuples.

The notion of formula domain independent
A formula is domain independent if its evaluation generates the same result even if the domains associated with attributes are extended by adding new values, not present in the initial database.

A syntactic condition has been introduced, referred to as safety, to assure such property.

The main idea of safety: we restrict our queries to queries the results of which only depend on the values present in the initial database.

The indepency of a formula from the domain is not decidable.

The safety condition is a syntactic condition sufficient to guarantee the independency of a formula from the domain.

We do not see this condition.

The expression \( \{ t:U_R \mid \neg t \in R \} \) is not safe.

The safe TRC and the relational algebra have the same expressive power.

The translation from one formalism to the other can be executed with a time complexity which is polynomial in the expression dimension.
Why two formalisms?

- Relational algebra
  - Procedural language
  - Useful for the system
- Relational calculus
  - Declarative language
  - Useful for the user
- Real query languages are based on the calculus, even though they adopt a user-friendly notation
- Query optimization is based on the relational algebra properties

Example

Select the names of the employees working in division D1

TRC
\[ \{ t : \{ \text{Name} \} \mid \exists s (s \in \text{Employees} \land s.\text{Name} = t.\text{Name} \land \exists u (u \in \text{Departments} \land s.\text{Dept#} = u.\text{Dept#} \land u.\text{Division} = \text{"D1"})) \} \]

Relational Algebra

(a) \( \Pi_{\text{Name}} (\sigma_{\text{Division}=\text{D1}} (\text{Employee } \bowtie \text{ Departments})) \)

Because the attribute Division belongs to the schema of Departments we can anticipate the selection, that is, we can re-write the query (a) as follows:

(b) \( \Pi_{\text{Name}} (\text{Employee } \bowtie \sigma_{\text{Division}=\text{D1}} (\text{Departments})) \)

Suppose that:
- The Employees relation contains \( n_1 \) tuples
- The Departments relation contains \( n_2 \) tuples (it is reasonable to assume that \( n_2 \leq n_1 \))
- The number of departments of division D1 are \( n_3 \) (\( n_3 \leq n_2 \))

Costs:
- (a) join: \( n_1 \times n_2 \), selection: \( n_1 \), projection: \( n_1 \), total: \( n_1 \times n_2 + 2 \times n_1 \)
- (b) selection: \( n_2 \), join: \( n_1 \times n_3 \), projection: \( n_1 \), total: \( n_1 \times n_3 + n_1 + n_2 \)

Cost(a) > Cost(b)

The system chooses strategy (b). Such strategy is very well known and it is called "push selections down"