The Relational Algebra and Relational Calculus

5DV119 — Introduction to Database Management
Umeå University
Department of Computing Science

Stephen J. Hegner
`hegner@cs.umu.se`
`http://www.cs.umu.se/~hegner`

The Roots of SQL

- It can scarcely be said that SQL has a clean and simple design.
- Rather, SQL is based upon the blending of many ideas, and has evolved over a long period of time.
- Nevertheless, SQL has its roots in two ideal query languages.

Relational Algebra: A *procedural language* grounded in basic operations on relations.

  - Widely used in algorithms for query optimization.

Relational Calculus: A *declarative language* grounded in first-order predicate logic.

- To understand better the capabilities and limitations of SQL, as well as for other reasons, it is therefore useful to study these two languages.
- They are part of almost all basic courses on database management given by faculties of science and technology at the university level.

Overview of the Relational Algebra

- The relational algebra is a *procedural* query language on relations.
- Its basic operations have one of the following forms:

$$\begin{aligned} \text{Relation} &\longrightarrow \text{Relation} \\ \text{Relation} \times \text{Relation} &\longrightarrow \text{Relation} \end{aligned}$$

- It therefore provides a basic computational model of how queries in SQL may be evaluated by a DBMS.
- It is often used in the internal representation of queries for the query optimizer in real relational DBMSs.
- A basic knowledge of the relational algebra can thus be very helpful in understanding why certain query operations are very expensive (in terms of time and computational resources) relative to others.

Overview of the Operations of the Relational Algebra

- The relational algebra is defined in terms of three kinds of operations on relations:

Operations specific to relations:

Projection: Relation → Relation: Trim some columns from a relation.

Selection: Relation → Relation: Trim some rows from a relation.

Join: Relation × Relation → Relation: Combine two relations by matching values.

The three fundamental set-theoretic operations: all Relation × Relation → Relation

Union: $X \cup Y =$ all elements in either X or Y.

Intersection: $X \cap Y =$ all elements in both X and Y.

Difference: $X \setminus Y$ or $X - Y =$ all elements in X which are not in Y.

A special operation of the form Relation → Relation:

Attribute renaming: Change the names of some attributes of a relation.

Projection

- The projection operation takes a "vertical" slice of a relation by dropping some columns while retaining others.
- The *projection* operator is represented by the lowercase Greek letter π, with the subscript identifying the columns to be retained.

$$\pi_{\{A_1,A_2,\ldots,A_k\}}(R)$$

- The semantics of this expression are exactly those of the following SQL query.

```
SELECT DISTINCT A1, A2, ..., Ak
FROM   R;
```

- This is a formal operation on sets; duplicates are not part of the model.
- Often, the set brackets are dropped in the subscript.

$$\pi_{A_1,A_2,\ldots,A_k}(R)$$

- If the attribute names are single letters, even the commas are sometimes dropped.

$$\pi_{A_1 A_2 \ldots A_k}(R)$$

Selection

- The selection operation takes a “horizontal” slice of a relation by dropping some rows while retaining others.
- The *selection* operator is represented by the lowercase Greek letter σ, with the subscript containing an expression which identifies the rows to be retained.

$$\sigma_{\varphi}(R)$$

- The semantics of this expression are exactly those of the following SQL query.

```
SELECT DISTINCT *
FROM   R
WHERE  φ;
```

- The expression φ is often written in a more formal, logical style than that used by SQL.

Example:

$$\sigma_{((\mathrm{DNo}=5)\wedge(\mathrm{Salary}\geq 30000))}(R)$$

Combining Expressions in the Relational Algebra

- The operations in the relational algebra themselves produce relations as results.
- Therefore, they may be composed.

Example: $\pi_{A_1,A_2,\ldots,A_k}(\sigma_{\varphi}(R))$ has the same meaning as

```
SELECT DISTINCT A1, A2, ..., Ak
FROM     R
WHERE    φ;
```

- Typing rules must be observed, since it is the composition of two distinct operations.

Example: While

$$\pi_{\mathrm{LName,SSN}}(\sigma_{\mathrm{Salary}\geq 30000}(\mathrm{Employee}))$$

makes perfect sense,

$$\sigma_{\mathrm{Salary}\geq 30000}(\pi_{\mathrm{LName,SSN}}(\mathrm{Employee}))$$

does not.

Assignment Programs in the Relational Algebra

- Instead of composing operations in functional notation, queries in the relational algebra may be expressed as a sequence of assignment statements.

Example: The functional composition

$$\pi_{\mathsf{LName,SSN}}(\sigma_{\mathsf{Salary}\geq 30000}(\mathsf{Employee}))$$

may also be expressed as the program of assignments

$$\begin{aligned} X_1 &\longleftarrow \sigma_{\mathsf{Salary}\geq 30000}(\mathsf{Employee}) \\ X_2 &\longleftarrow \pi_{\mathsf{LName,SSN}}(X_1) \end{aligned}$$

with X_2 as the final result.

- It is often easier to read and follow such sequence of assignments than to read and follow a complex functional composition.

Join

- The join is a binary operation represented by the "bowtie" symbol $\bowtie$.
- It is basically the inner join of SQL.
- There are, however, a number of variants depending upon the subscript (or lack thereof).
- The expression

  $$R_1 \bowtie_\varphi R_2$$

  has the semantics of the SQL expression

  ```
  SELECT *
  FROM   R_1 JOIN R_2 ON (φ);
  ```

  provided φ is represented in the correct way.

Example: Employee $\bowtie_{(\mathrm{DNo}=\mathrm{DNumber})}$ Department

has the meaning of

```
SELECT *
FROM   Employee JOIN Department ON (DNo=DNumber);
```

Further Join Conventions

- Multiple conditions may be shown in various ways:

$$\text{Employee} \bowtie_{(\text{DNo}=\text{DNumber})\wedge(\text{Super\_SSN}=\text{Mgr\_SSN})} \text{Department}$$

$$\text{Employee} \bowtie_{\{(\text{DNo}=\text{DNumber}),(\text{Super\_SSN}=\text{Mgr\_SSN})\}} \text{Department}$$

$$\text{Employee} \bowtie_{(\text{DNo}=\text{DNumber}),(\text{Super\_SSN}=\text{Mgr\_SSN})} \text{Department}$$

- These all have the meaning of

```
SELECT *
FROM   Employee JOIN Department
                ON ((DNo=DNumber) AND (Super_SSN=Mgr_SSN));
```

- Other logical connectives:

$$\text{Employee} \bowtie_{(\text{DNo}=\text{DNumber})\vee(\text{Super\_SSN}=\text{Mgr\_SSN}))} \text{Department}$$

has the meaning of

```
SELECT *
FROM   Employee JOIN Department
                ON ((DNo=DNumber) OR (Super_SSN=Mgr_SSN));
```

but is not a construction which occurs often in practice.

Natural and Cross Joins

- The natural join is indicated by the absence of any subscripts on $\bowtie$.
- ☡ The textbook uses the symbol $*$ for natural join, although this notation is rather dated (and was used to denote inner join in early literature).
- Thus, the following two expressions are equivalent.

$$\text{Department} \bowtie \text{Dept\_Locations}$$

$$\text{Department} \; * \; \text{Dept\_Locations}$$

with the same meaning as

```
SELECT *
FROM   Department NATURAL JOIN Dept_Locations;
```

- Note that $\bowtie_{\emptyset}$ is the *cross join*, with no matches. ($\emptyset = \{\} =$ empty set.)
- Thus, Department $\bowtie_{\emptyset}$ Dept_Locations has the meaning of

```
SELECT *
FROM   Department JOIN Dept_Locations ON (TRUE);
```

- This cross join (or *Cartesian product*) is also denoted

$$\text{Department} \times \text{Dept\_Locations}.$$

Theta Join

- Theta joins may be specified in the relational algebra in the obvious way.

Query: Find those employees who have an older dependent.

$$\text{Employee} \bowtie_{(\text{SSN}=\text{ESSN}) \wedge (\text{Employee.BDate} > \text{Dependent.BDate})} \text{Dependent}$$

is equivalent to:

```
SELECT DISTINCT LName, FName, MInit, SSN
FROM   Employee JOIN Dependent
                ON ((SSN=ESSN)
                AND (Employee.BDate > Dependent.BDate));
```

- which is equivalent to:

```
SELECT DISTINCT LName, FName, MInit, SSN
FROM   Employee JOIN Dependent ON (SSN=ESSN)
WHERE  (Employee.BDate > Dependent.BDate);
```

Renaming

- Recall that it is sometimes necessary to have multiple copies of the same relation.

Query: Find the name of the supervisor of each employee.

```
SELECT E.LName, E.FName, E.MInit, S.LName, S.FName, S.MInit
FROM   Employee AS E JOIN Employee AS S
                         ON (E.Super_SSN=S.SSN);
```

- In the relational algebra, there is a *rename* operation for this.
- There are two main formats:
  - $\rho_{R'}(R)$ returns a copy of R named R', with the same attribute names.
  - $\rho_{R'(A'_1,A'_2,\ldots,A'_k)}(R)$ returns a copy of R named R', with the the attributes renamed to $A'_1, A'_2, \ldots, A'_k$.
- Name qualifiers are used as in SQL.
- However, the original relation does not require a qualifier.

Renaming Examples

Query: Find the name of the supervisor of each employee.

- The above query as a sequence of steps in the relational algebra, with X_3 the answer, using each of the renaming conventions:

$$X_1 \longleftarrow \rho_{\mathsf{E}}(\mathsf{Employee})$$

$$X_2 \longleftarrow \rho_{\mathsf{S}}(\mathsf{Employee})$$

$$X_3 \longleftarrow X_1 \bowtie_{(\mathsf{E.Super\_SSN=S.SSN})} X_2$$

$$X_4 \longleftarrow \pi_{\mathsf{E.LName,E.FName,E.MInit,S.LName,S.FName,S.MInit}}(X_3)$$

$$X_1 \longleftarrow \rho_{\mathsf{S(FName',MInit'.LName',SSN',BDate',Address',Sex',Salary',Super\_SSN',DNo')}}(\mathsf{Employee})$$

$$X_2 \longleftarrow \mathsf{Employee} \bowtie_{(\mathsf{Super\_SSN=SSN'})} X_1$$

$$X_3 \longleftarrow \pi_{\mathsf{LName,FName,MInit,LName',FName',MInit'}}(X2)$$

Another Renaming Example

Query: Find the Name and SSN of those employees who work on exactly one project.

- The above query as a sequence of steps in the relational algebra, with X_7 the answer:

$X_1 \longleftarrow \rho_{\mathrm{W}}(\text{Works\_On})$ -- Copy of Works_On

$X_2 \longleftarrow \text{Works\_On} \bowtie_{(\mathrm{PNo} \neq \mathrm{W.PNo}) \wedge (ESSN = W.ESSN)} X_1$

$X_3 \longleftarrow \rho_{X_a(SSN)}(\pi_{\mathrm{ESSN}}(X_2))$ -- Employees who work on >1 projects

$X_4 \longleftarrow \pi_{\mathrm{SSN}}(\text{Employee}) \setminus \rho_{X_b(SSN)}(\pi_{\mathrm{ESSN}}(\text{Works\_On}))$ -- Employees who work on <1 projects

$X_5 \longleftarrow \pi_{\mathrm{SSN}}(\text{Employee}) \setminus (X_3 \cup X_4)$ -- Employees who work on $=1$ project

$X_6 \longleftarrow X_5 \bowtie \text{Employee}$

$X_7 \longleftarrow \pi_{\mathrm{LName,FName,MInit,SSN}}(X_6)$ -- Add the names

Set Operations

- The following set operations are considered part of the relational algebra:

  Union: $X \cup Y$ = all elements in either X or Y.

  Intersection: $X \cap Y$ = all elements in both X and Y.

  Difference: $X \setminus Y$ or $X - Y$ = all elements in X which are not in Y.

- They may only be applied when the elements in each set are of the same type.
  - If they are tuples, they have the same number of columns.
  - The attributes for matching columns must be of the same type.

Recall Division in SQL

- The division operation has already been seen in the following SQL example:

Query: Find all employees who work on every project which Alicia Zeyala (999887777) also works on. Exclude Alicia herself.

Recall the strategy: Find all employees E for which there is no project P which Alicia works on but E does not work on.

```
SELECT DISTINCT LName, FName, MInit, SSN
FROM Employee JOIN Works_On ON (SSN=ESSN)
WHERE NOT EXISTS (SELECT PNo
                  FROM   Works_On
                  WHERE  (ESSN='999887777')
                  EXCEPT (SELECT PNo
                          FROM   Works_On
                          WHERE  (SSN=ESSN)))
                AND (SSN<>'999887777');
```

- This operation may be formalized within the relational algebra.

Formalization of Division via Example

Works_On

ESSN	PNo

PList

PNo

- Consider the schema as shown to the right.

Query: Find the SSNs of those employees in Works_On who work on every project in PList.

- Here is an assignment program in the relational algebra which provides a solution:

$X_1 \longleftarrow \pi_{\mathrm{ESSN}}(\mathrm{Works\_On})$ -- Workers: employees who work on some project

$X_2 \longleftarrow X_1 \times \mathrm{PList}$ -- Every worker works on every project in PList

$X_3 \longleftarrow X_2 \setminus \mathrm{Works\_On}$ -- The “Does_Not_Work_On” relation

$X_4 \longleftarrow \pi_{\mathrm{ESSN}}(X_3)$ -- Workers who do not work on some project in PList

$X_5 \longleftarrow X_1 \setminus X_4$ -- Employees who work on every project in PList

- As a single expression:

$$\pi_{\mathrm{ESSN}}(\mathrm{Works\_On}) \setminus (\pi_{\mathrm{ESSN}}(\pi_{\mathrm{ESSN}}(\mathrm{Works\_On}) \times \mathrm{PList}) \setminus \mathrm{Works\_On})$$

Formalization of Division

R

<u>A</u>	<u>B</u>

S

<u>B</u>

Query: Find the A's in R which are associated with every B in S.

- Here is an assignment program in the relational algebra which provides a solution:

$$\begin{array}{lll} X_1 \longleftarrow \pi_A(R) & & \text{-- all } A\text{'s} \\ X_2 \longleftarrow X_1 \bowtie S & & \text{-- } A \times B \\ X_3 \longleftarrow X_2 \setminus R & & \text{-- } (A \times B) \setminus R \\ X_4 \longleftarrow \pi_A(X_3) & & \text{-- } A\text{'s not associated with some } B \\ X_5 \longleftarrow X_1 \setminus X_4 & & \text{-- } A\text{'s associated with every } B \end{array}$$

- As a single expression:

$$\pi_A(R) \setminus (\pi_A(\pi_A(R) \bowtie S) \setminus R)$$

- This division is written $R \div S$.
- This extends easily to $R[\mathbf{A}]$, $S[\mathbf{B}]$, with sets $\mathbf{A}$, $\mathbf{B}$ of attributes satisfying $\mathbf{B} \subseteq \mathbf{A}$.

Additional Operations of the Relational Algebra

- Many additional operations may be added to the relational algebra to make it as powerful as SQL, including:
    - Aggregation and grouping operators
    - Outer join
    - Recursive closure operations

- These are relatively straightforward to define, but will not be pursued further in this course.

Declarative Query Languages

Procedural: A query language is *procedural* if it indicates explicitly how to compute its result.

- The relational algebra is a procedural query language.

Declarative: A query language is *declarative* if it indicates *what* to compute without requiring any indication of *how*.

- A great advantage of the relational model of data is that it admits a fully declarative query language.
- This means that the query language may be decoupled completely from the procedural model of computation.
  - This is particularly important for the support of non-technical users.
- For other data models, including object-oriented models, such a decoupling is difficult, if possible at all.
- The declarative query language for the relational model is called the *relational calculus*, and will be examined briefly.

Propositional Logic

- The relational calculus is based upon first-order mathematical logic, which in turn is based upon propositional logic.

- Familiarity with propositional logic is assumed, including:

Connectives: $\vee$, $\wedge$, $\neg$, $\Rightarrow$.

  - $(A \Rightarrow B)$ is *defined* to mean $((\neg A) \vee B)$.

Well-formed formulas (WFFs): $(A \wedge ((\neg B) \vee C) \Rightarrow (D \vee E))$

DeMorgan's Laws:

$$(\neg(A \wedge B)) = ((\neg A) \vee (\neg B))$$
$$(\neg(A \vee B)) = ((\neg A) \wedge (\neg B))$$

The Tuple Relational Calculus

- The specific relational calculus presented here is called the *tuple relational calculus*.

Tuple variables: The tuple relational calculus works with *tuple variables*.

- Each tuple variable has a *type* which is one of the relations in the schema.
  - $R(t)$ declares tuple t to be of type R.

  Example: Employee(e).

- The value for a specific attribute is retrieved using standard notation.
  - $t.A$ retrieves the A-value of tuple variable t.

  Example: e.Salary.

  - Call an expression such as $t.A$ a *tuple-field variable*.

- For those familiar with first-order predicate logic, each $t.A$ corresponds (roughly) to a variable.

The Tuple Relational Calculus — 2

Quantifiers: Quantifiers are used in expressions in the calculus.

$\forall$: For all.

$\exists$: There exists.

- Queries are of the form

$$\{t_1.A_1, t_2.A_2, \ldots, t_k.A_k \mid \varphi\}$$

  in which:

  - Each $t_i.A_i$ is a tuple-field variable.
  - φ is a logical formula in which exactly the elements of $\{t_1, t_2, \ldots, t_k\}$ are *free* (not within the scope of any quantifier).

- Rather than present a long formal syntax of well formedness, a number of examples will be used to illustrate the various constructions and techniques.

Examples in the Tuple Relational Calculus

Query: Find the name and SSN of those employees who work on some project.

$$\{e.\mathrm{LName}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e) \\ \wedge (\exists w)(\mathrm{Works\_On}(w) \wedge (e.\mathrm{SSN} = w.\mathrm{ESSN}))\}$$

Query: Find the name and SSN of those employees who work on the ProductX project.

$$\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e) \\ \wedge (\exists w)(\exists p)(\mathrm{Works\_On}(w) \wedge \mathrm{Project}(p) \wedge (p.\mathrm{PName} = \text{'ProductX'}) \\ \wedge (p.\mathrm{PNumber} = w.\mathrm{PNo}) \wedge (e.\mathrm{SSN} = w.\mathrm{ESSN}))\}$$

Query: Find the name and SSN of those employees who work on every project.

$$\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e) \\ \wedge (\forall p)(\mathrm{Project(p)} \Rightarrow \\ (\exists w)(\mathrm{Works\_On}(w) \wedge (e.\mathrm{SSN} = w.\mathrm{ESSN}) \wedge (p.\mathrm{PNumber} = w.\mathrm{PNo})))\}$$

- Note how easy and natural division is in the tuple relational calculus!

Examples in the Tuple Relational Calculus — 2

Query: Find the name and SSN of those employees who work on exactly one project.

$$\begin{aligned}&\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e)\\ &\quad \wedge (\exists w)(\mathrm{Works\_On}(w) \wedge (e.\mathrm{SSN} = w.\mathrm{ESSN}))\\ &\qquad \wedge (\forall w_1)(\forall w_2)((\mathrm{Works\_On}(w_1) \wedge \mathrm{Works\_On}(w_2) \wedge (w_1.\mathrm{ESSN} = w_2.\mathrm{ESSN})\\ &\qquad\qquad \wedge (w_1.\mathrm{ESSN} = e.\mathrm{SSN})) \Rightarrow (w_1.\mathrm{PNo} = w_2.\mathrm{PNo}))\}\end{aligned}$$

Query: Find the name and SSN of those employees who do not work on any project.

$$\begin{aligned}&\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e)\\ &\qquad \wedge (\neg(\exists w)(\mathrm{Works\_On}(w) \wedge (e.\mathrm{SSN} = w.\mathrm{ESSN})))\}\end{aligned}$$

or

$$\begin{aligned}&\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e)\\ &\qquad \wedge ((\forall w)(\mathrm{Works\_On}(w) \Rightarrow (e.\mathrm{SSN} \neq w.\mathrm{ESSN})))\}\end{aligned}$$

Examples in the Tuple Relational Calculus — 3

Query: Find the name and SSN of those employees who work on at least two distinct projects.

$$\begin{aligned}\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e)\\ \wedge (\exists w_1)(\exists w_2)((\mathrm{Works\_On}(w_1)\wedge\mathrm{Works\_On}(w_2) \wedge (w_1.\mathrm{ESSN} = w_2.\mathrm{ESSN})\\ \wedge (w_1.\mathrm{ESSN} = e.\mathrm{SSN})) \wedge (w_1.\mathrm{PNo} \neq w_2.\mathrm{PNo}))\}\end{aligned}$$

Query: Find the name and SSN of those employees who work on exactly two distinct projects.

$$\begin{aligned}\{e.\mathrm{Lname}, e.\mathrm{FName}, e.\mathrm{MInit}, e.\mathrm{SSN} \mid \mathrm{Employee}(e)\\ \wedge (\exists w_1)(\exists w_2)((\mathrm{Works\_On}(w_1)\wedge\mathrm{Works\_On}(w_2) \wedge (w_1.\mathrm{ESSN} = w_2.\mathrm{ESSN})\\ \wedge (w_1.\mathrm{ESSN} = e.\mathrm{SSN})) \wedge (w_1.\mathrm{PNo} \neq w_2.\mathrm{PNo}))\\ \wedge (\forall w_1)(\forall w_2)(\forall w_3)(\mathrm{Works\_On}(w_1) \wedge \mathrm{Works\_On}(w_2) \wedge \mathrm{Works\_On}(w_3)\\ \wedge (w_1.\mathrm{ESSN} = w_2.\mathrm{ESSN})\wedge(w_1.\mathrm{ESSN} = w_3.\mathrm{ESSN}) \wedge (w_1.\mathrm{ESSN} = e.\mathrm{SSN}))\\ \Rightarrow ((w_1.\mathrm{PNo} = w_2.\mathrm{PNo}) \vee (w_1.\mathrm{PNo} = w_3.\mathrm{PNo}) \vee (w_2.\mathrm{PNo} = w_3.\mathrm{PNo}))\}\end{aligned}$$

Remarks about Queries in the Relational Calculus

☡ $(\neg\forall)$ and $(\neg\exists)$ are ambiguous and incorrect, and should never be used.

  Question: What does $(\forall e)(\neg\exists w)$ mean?

  - Write $(\forall e)\neg(\exists w)$ if that is what is meant.

- Recall that negation “flips” quantifiers.
  - $\neg((\forall x)(\varphi))$ is equivalent to $(\exists x)((\neg\varphi))$.
  - Think about it for simple examples.
  - Similarly $\neg((\exists x)(\varphi))$ is equivalent to $(\forall x)((\neg\varphi))$.
- Keep in mind that $\varphi_1 \Rightarrow \varphi_2$ is <u>defined</u> to mean $(\neg\varphi_1)\vee\varphi_2$.
- The value of a variable must always be defined in one of two ways.
  - By nature of lying within the scope of a quantifier
    - $(\forall e)(\exists w)$(e and w are bound here.).
  - By nature of being in the argument list of a query.
    - e and w arguments in $\{e.A, w.B \mid \text{Employee}(e)\wedge\text{Works\_On}(w)\wedge\langle\text{some formula}\rangle\}$.
- The type of each variable in the argument list must be defined in the formula of the query.

The Expressive Power of the Algebra and Calculus

- A major, nontrivial result is the following:

Theorem: The relational algebra and the tuple relational calculus have the same expressive power. □

- This means that there is no loss of expressive power in using an entirely declarative language for querying relational databases.
- There has been substantial debate, with supporting research for both sides, for the relative merits of declarative query languages versus procedural query languages.
- In any case, SQL is blend of the two, with choices made for historical rather than scientific reasons.
- Still, it is very useful to be aware of the distinction between these two flavors of query expression.

Safe Queries

Theorem: The relational algebra and the tuple relational calculus have the same expressive power. □

- There is one restriction which must be imposed for this result to hold: the queries must be *safe*.
- Roughly speaking, a query is safe it can only return answers whose attribute values occur in the database being queried.

Example of an unsafe query: Give the set of all numbers which are not the salary of some employee.

- A query in the relational algebra is always safe.
- A query in the tuple relational calculus is guaranteed to be safe if every tuple variable in the argument list is bound to a type.
  - This is guaranteed in the formalism which has been developed here.
- Unsafe queries can arise in an alternative called the *domain relational calculus*, which is essentially standard first-order logic.
- The domain calculus will not be considered here.